The Hierarchical Subspace Iteration Method for Computing Vibration Modes of Elastic Objects

Master's Thesis

Julian van Dijk

The Hierarchical Subspace Iteration Method for Computing Vibration Modes of Elastic Objects

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Julian van Dijk
born in Reeuwijk, the Netherlands

Computer Graphics and Visualization Group
Department of Intelligent Systems
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

The Hierarchical Subspace Iteration Method for Computing Vibration Modes of Elastic Objects

Author: Julian van Dijk
Student id: 4695283
Email: j.m.c.vandijk@student.tudelft.nl

Abstract

The Hierarchical Subspace Iteration Method is a novel method used to compute eigenpairs of the Laplace-Beltrami problem. It reduces the number of iterations required for convergence by restricting the problem to a smaller space and prolonging the solution as a starting point. This method has shown great performance improvements for Laplace-Beltrami eigenproblems. We propose an adaptation to the Hierarchical Subspace Iteration Method that allows for computing vibration modes of elastic objects. We evaluate potential optimizations that can be made, as well as the performance characteristics of the method. Our method was shown to be faster than SIM in most cases while even beating Matlab's Lanczos solver in some cases.

Thesis Committee:

Chair:	Prof. Dr. E. Eisemann, Faculty EEMCS, TU Delft
University supervisor:	Dr. K Hildebrandt, Faculty EEMCS, TU Delft
Committee Member:	Dr. E. Isufi, Faculty EEMCS, TU Delft

Acknowledgements

To start, I would like to thank my thesis supervisor Dr. Klaus Hildebrandt who guided me during the entire project. I have never met a person who has such a dire need for more office space to accommodate all the formulas and explanations that are written on the walls. When I could not grasp a subject you would explain it to me in a clear step-wise manner (with the necessary ”wiping the wall clean” breaks when necessary). In addition, I would like to thank Prof. Dr. Elmar Eisemann and Dr. Elvin Isufi for taking the time to read, understand and provide feedback on my work.

I would like to express gratitude to Dr. Ahmad Nasikun who was so kind as to share the original code of the HSIM paper as well as introduce me to its inner workings over a Zoom call at inappropriate times for both timezones. That introduction saved me weeks of possible debugging and allowed me to get used to all the parameters rather quickly. Thank you.

Special gratitude goes out to my parents, my brothers and sisters, my friends and especially Merel who supported me throughout this long process and gave direction to my thoughts when I lost steering.

Preface

I can fondly remember the start of my studies. I was thrilled, excited and healthily fearful of all that would come. Computers have always been my ”thing” and this study continuously confirmed that, but with each amount of ECTs I achieved I also felt that one task that scared me most came closer: ”Writing a thesis”. I could hardly write a coherent 2 pager in high school after staring at the wall for weeks which made this fear not fully grounded.

However, those fears have aged as of now and things have changed. Here I am typing a document's preface with a page count in the two-digit range. Something that would have seemed impossible for me 6 years ago. I learned a lot during those years which made me feel ready to tackle this project on day one, but I could not have guessed that I had so much more to learn.

I fondly remember the first meeting with both Klaus and Elmar where I proposed the topic I wanted to work on. Klaus proposed two possible research directions and I chose the direction that looked the hardest for me. During the meeting, Elmar asked if this topic was my choice to which I happily replied with: ”Yes”. Looking back at it now I understand the question: Which master student starts working on an eigensolver voluntarily? This thought sprung up more and more when I tried to explain my topic to my peers. The moment I said ”Eigen” they would tune out.

This choice of topic was a strategic one, however. I am quite competitive from character and this research could show strong gains in compute time where we could beat the status quo algorithms that are used today. I liked the idea of trying to optimise the implementation for the specific problem and possibly winning from Lanczos' algorithm. The matrices generated by the linear elasticity problem exposed some weaknesses in the HSIM code when decreasing sparsity. I can attest that ironing those out one by one was a truly satisfying experience.

Julian van Dijk
Delft, the Netherlands
November 17, 2023

Contents

Chapter 1

Introduction

Engineers and artists have become more and more acquainted with digital modelling and simulation within their work. Simulating real-world physics allows the computer to do the heavy lifting while also preventing human mistakes. A known simulation problem is the calculation of vibration modes. Vibration modes show the natural frequencies of objects. These frequencies are of great importance when designing real-world structures. If the structure resonates with a frequency in its vicinity, it might become unstable. A good example of this behaviour is the Tacoma Narrows Bridge incident recorded by Gaal and Levine [2016].

Vibration modes have a multitude of other applications, such as audio simulation, antenna design and animation. Our work will focus on the calculation of low-frequency vibration modes of elastic objects. Calculating the vibration modes of a given volumetric mesh requires us to dive into the Finite Element Method (FEM). The finite element method allows us to perform a localized elasticity calculation on the tetrahedrons of the mesh. The results of the calculation can then be used to build a large sparse stiffness matrix detailing the interaction of the vertices of the mesh. We can then use this stiffness matrix in combination with a mass matrix to find the vibration modes.

Our proposed solution applies the Hierarchical Subspace Iteration Method (HSIM) by Nasikun and Hildebrandt [2022] to a volumetric mesh to speed up vibration mode calculation for the low to mid-frequency spectrum. the HSIM method has already been applied to Laplace-Beltrami Eigenproblems on surfaces, showing significant performance improvements. We researched the application and possible performance improvements for the vibration modes eigenproblem of our solution.

The adaptation will also introduce a completely different problem to test on HSIM. The global stiffness matrix will be more dense. The average vertex valence of a tetrahedral mesh is higher than the average vertex valence of a triangle mesh [Gumhold et al., 1999]. The other contribution to matrix density is the degrees of freedom per vertex. Our stiffness matrix will have 3 degrees of freedom per vertex, where the degrees of freedom represent the 3 orthogonal axes of movement in 3D space. These two factors create a denser eigenproblem with a different structure than Laplace-Beltrami Eigenproblems. The increased density will require more operations for prolongation, hierarchy creation and factorization. Possibly causing extra cost for the hierarchy approach.

Another difference between Laplace-Beltrami eigenproblems and elasticity problems is the distribution of the eigenvalues. Laplace-Beltrami eigenproblems have relatively more closely distributed eigenvalues than elasticity problems. An example of the eigenvalue distribution of Laplace-Beltrami on a sphere can be seen in Figure 9 in [Nasikun and Hildebrandt, 2022]. The eigenvalues range between 0 and 250 for the lowest 250 eigenvalues. A volumetric sphere mesh with 182k degrees of freedom has its 250 lowest eigenvalues ranging between 1803 to $13.7 * 10e^9$. This stark difference is also reflected in the condition number, which is often significantly higher for elasticity problems.

It is possible that HSIM will provide a computational speedup compared to SIM for the vibration modes problem. The computational speed of HSIM grew significantly when calculating a larger amount of eigenpairs in the original paper. This gain in computational speed might translate to our problem as well.

1.1 Research question(s)

The main research question investigated in this thesis is:

How can the HSIM be applied to solve generalized Eigenvalue problems for the vibration modes of elastic meshes?

- How can the HSIM be applied to isotropic elasticity problems
- What adaptations can be made to the HSIM method to better fit the generalized Eigenvalue problem used for calculating vibration modes?
- What are the optimal parameters to use when applying the HSIM to elasticity problems?

Chapter 2

Related work

Solving large sparse eigenproblems is a well-explored domain in computer science, mathematics and structural engineering. This chapter discusses the methods that are available at the time of writing. In this chapter we will focus on eigensolvers that apply to our elasticity problem.

2.1 Modal analysis

Modal analysis, also known as vibration analysis, is the study of vibration characteristics of a structure. It is often used together with the finite element method to determine structural characteristics, the natural frequencies and mode shapes of a system. These methods are used within multiple engineering fields as shown in these examples: [Guo et al., 1992], [LU and ABBOTT, 1996] and [Zheng and Kessissoglou, 2004]. Research on objects and their materialistic properties has already been done and published, creating a large availability of computed data. However, the need to compute more modes, using larger models or entirely new simulation models, is still present. As can be seen in recent research about axial vibration analysis of embedded love-bishop nanorods by Ömer Civalek and Numanoğlu [2020] or the introduction of a new three-layer composite FEM model by Chi Tho et al. [2023].

Vibration modes are applicable to a multitude of problems. They can be used to characterize a model/structure, which can then be used as a comparison for real-life measurements aiding in determining structural integrity. Another application is that of prevention. Using simulated models, one can determine whether or not the modes would interfere with the application of the model. Preventing the accident that occurred at Tacoma Narrows Bridge incident recorded by Gaal and Levine [2016]. This application extends into aiding material and structural design. A good example is the technical report of Dean and Crocker [2023], where they use vibration analysis for designing new adhesives. Modal analysis is also used in conjunction with harmonic analysis by computing the modes of a model and the possible interactions with its surroundings as shown by Yan et al. [2023] or in this the work by Xu et al. [2023] where they allow fast tonal noise predictions of multi-rotor setups using modal analysis.

2.2 Lanczos algorithm

The Lanczos algorithm is an effective solver for large symmetric eigenproblems. The Lanczos algorithm is widely used for large symmetric eigenproblems. It is often used because of its superior speed of convergence compared to SIM. The algorithm is prone to numerical instability due to floating point rounding errors, resulting in a loss of orthogonality in its basis vectors. The underlying issue is the Gram-Schmidt procedure Giraud et al. [2005] that is being used for the orthogonalization of the basis vectors. Accumulations of accuracy errors due to matrix-vector multiplications eventually result in a linearly dependent vector that causes instability. Implementations of the Lanczos algorithm often resolve this issue by orthogonalizing the linearly dependent vectors and then restarting the current iteration. A more detailed explanation of the Lanczos algorithm can be found in Cullum and Willoughby [2002]. The C++ library Arpack Lehoucq et al. [1997] has implementations for different variants of the Lanczos algorithm. Arpack offers an implementation of the Implicitly Restarted Lanczos method, which is used by Matlab[1]. However, the Lanczos method does have limitations preventing an even wider adoption. The Lanczos method is intrinsically hard to run in parallel [Bathe and Ramaswamy, 1980b]. There is, however, recent research available that suggests strong parallel performance improvements by Zbikowski and Johnson [2023].

2.3 Subspace iteration method

The subspace iteration method originally developed by Bathe and Wilson [1973] aims to aid computation for frequencies and mode shapes in structures. This method is numerically stable. Another interesting property is the subspace iteration itself. Every iteration performs an inverse iteration, which evolves the starting vectors towards the closest eigenpairs. Choosing starting vectors with a smaller distance to the desired eigenpairs results in less required iterations for convergence. It is thus very important to select good starting vectors. There has been a lot of research regarding the precomputation of starting vectors to use in the subspace iteration method. A few examples are composed in an article by Cheu et al. [1987a]. The subspace iteration method also profits from being able to be run in parallel with often achieving approximate linear scaling as shown by Bathe and Ramaswamy [1980b]. Other optimizations have been proposed, such as aggressive shift strategies Zhao et al. [2007] and applying turning vectors that steer the subspace faster towards a solution introduced by Kim and Bathe [2017] and extended by Wilkins [2019].

2.4 Turning vectors

During our search for optimization techniques, we were introduced to an extension of the Subspace Iteration method. This extension adds a few steps for every iteration that are meant to optimize the iteration vectors with turning vectors. This method is introduced by Kim and Bathe [2017]. The forward-turning vectors allow a speed-up in convergence at

[1] A programming and numeric computing platform https://mathworks.com/products/matlab.html

the cost of a slightly more intensive computation per iteration. The authors show that they can achieve significant speedups up to 3 to 5 times by reducing the number of iterations necessary for convergence compared to the SIM implementation specified by Bathe [2013]. However, all their testing has been done on a single-core laptop

Wilkins [2019] has proposed an extension of this method. The extension implements a second routine that allows turning the already turned turning vectors. The authors show that there is a small decrease in iterations compared to the previous enriched method in most test cases. This comparison is again made on a single CPU core. The authors do, however, address the sequential implementation in the conclusions part. Here, they state that the algorithm could benefit from further research mainly focused on a parallel implementation of the algorithm. The current execution order of the algorithm does not allow for a parallel implementation due to the sequential dependencies introduced by the turning step. We have implemented the enriched subspace iteration method next to our subspace iteration method used in HSIM to allow for comparison. In our tests, we see that the implementation does reduce the number of iterations. However, the implementation is heavily impacted by the number of needed turning vectors. If an iteration requires a lot of turning vectors, it will negatively impact the processor utilization of the algorithm as well as the real-world computation time of the iteration, but it will reduce the number of required iterations.

2.5 Approximation Techniques

An approximation technique produces results within a certain tolerance. These techniques are well suited for applications where strict accuracy is not always necessary. The subspace construction idea originated from an approximation technique as well. The paper Fast Approximation of Laplace-Beltrami Eigenproblems by Nasikun et al. [2018] introduces this method. The method is used to approximate Laplace-Beltrami eigenpairs by creating a prolongation and restriction operator that restricts the problem to a smaller subspace. The solution is then prolonged to the original size, resulting in an approximate solution with little computation time.

The authors discuss multiple applications where approximations of Laplace-Beltrami eigenpairs are useful, such as shape DNA Reuter et al. [2006] and the simulation of elastic deformables. Where they show a small average relative error during simulation. Approximate solutions are often used for visual applications where speed is preferred over accuracy within a specified tolerance. Another example of this application is the Vivace paper by Fratarcangeli et al. [2016], where they introduce a method of simulating stable soft body dynamics within frame time. Approximate solutions are well fitted to applications where memory, computing, or both are limited, and accuracy is not the main focus. An approximation method can also be used to precondition the subspace for the subspace iteration method where a well-conditioned subspace can speed up convergence as seen in [Bramble et al., 1996]

Chapter 3

Background

In the following section, we will provide the mathematical basis that is required for our problem statement. We start with the definition of linear elasticity and build up to our generalized eigenproblem, on which we will apply our method in the next chapter.

3.1 Linear elasticity

This section makes use of the course notes of Sifakis and Barbic [2012] as a basis for its introduction to this topic and the associated formulas. The course notes do not introduce any new methods or theories.

3.1.1 Deformation

Mesh deformation is a concept in the area of computer graphics. It is applied in a wide range of 3D media. Deforming a mesh is, however, not easily done by hand. Elastic deformations can be useful for geometric modelling tasks. A way to achieve these deformations is through simulation. Elastic objects want to return to their original shape with a certain amount of force. This force is often non-uniformly continuously divided over the object. For our purposes, we will have to localize the forces to each vertex of the mesh. This is where the classical FEM method comes in.
We can define the object to be in a coordinate-based space where we denote Ω as the volumetric domain occupied by the object. A deformation gradient tensor can be created for this domain $\mathbf{F} \in \mathbf{R}^{3x3}$. Where $\overrightarrow{X} = (X_1, X_2, X_3)^T$ and $\overrightarrow{\phi}(\overrightarrow{X}) = (\phi_1(\overrightarrow{X}), \phi_2(\overrightarrow{X}), \phi_3(\overrightarrow{X}))^T$. Where $\overrightarrow{\phi}(\overrightarrow{X})$ is the *deformation function* of the given deformation and $\overrightarrow{X}$ is the initial position.

$$\mathbf{F} = \frac{\partial \phi}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial \phi_1}{\partial X_1} & \frac{\partial \phi_1}{\partial X_2} & \frac{\partial \phi_1}{\partial X_3} \\ \frac{\partial \phi_2}{\partial X_1} & \frac{\partial \phi_2}{\partial X_2} & \frac{\partial \phi_2}{\partial X_3} \\ \frac{\partial \phi_3}{\partial X_1} & \frac{\partial \phi_3}{\partial X_2} & \frac{\partial \phi_3}{\partial X_3} \end{bmatrix}$$

An elastic mesh is capable of storing energy in the form of strain energy. This energy is aimed at returning the mesh to its original shape. When working with hyperelastic material,

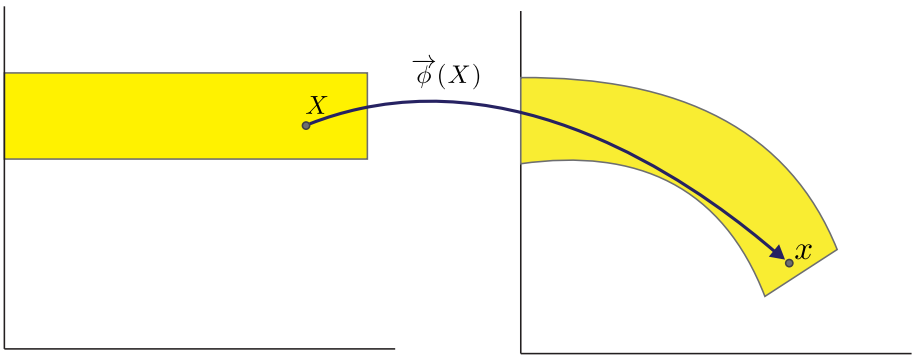


Figure 3.1: An Illustration of the deformation function for a bending bar.

one can derive energy from the original shape and the deformed shape. This does not work for other material types, such as viscoelastic materials, where the path of the deformation plays a role in the resulting forces.

The course notes of Sifakis and Barbic [2012] discuss that it is possible to derive an energy density function that takes the local deformation gradient as input and returns the energy density. These transformations require a few steps.

3.1.2 Strain measure

The Green strain tensor:

$$E = \frac{1}{2}(F^T F - I) \tag{3.1}$$

This tensor defines the strain energy in the system defined by the deformation gradient. If the system is in its rest state $F = I$, it results in $E = 0$. The same would hold true for pure rotations. In that case $F^T F = I$, which would result in $E = 0$. Sifakis and Barbic [2012] argue that one can decompose the deformation gradient into a rotation matrix and a symmetric matrix: $F = RS$. The rotation matrix is equal to the identity matrix when multiplied with its transpose. This allows us to write down the green strain tensor as:

$$E = \frac{1}{2}(S^2 - I) \tag{3.2}$$

This, however, shows a property of the green strain tensor. It is a quadratic function of deformation. A quadratic function will result in nonlinear functions. Therefore, Sifakis and Barbic [2012] discuss another tensor derived through a Taylor expansion around the undeformed configuration:

$$\varepsilon = \frac{1}{2}(F + F^T) - I \tag{3.3}$$

Called the small strain tensor. This tensor allows for a lightweight model: Linear elasticity.

3.1.3 Energy density function

An energy density function describes the properties of the material and its internal dependencies. For our solution, we will make use of the Linear Elasticity Model since we look at small vibrations around the rest configuration where the linear component is dominant. This model takes the small strain tensor (derived from the deformation gradient) as input together with two inputs λ, μ called the *Lamé coefficients*. These coefficients are derived from the two material properties: Young's modulus[1] k and Poisson's ratio[2] v. Which results in the strain energy density function:

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \qquad \mu = \frac{E}{2(1+\nu)} \tag{3.4}$$

$$\Psi(F) = \mu\varepsilon : \varepsilon + \frac{\lambda}{2} tr^2(\varepsilon) \tag{3.5}$$

3.2 Finite Element Methods

It is now possible for us to calculate the strain energy for a given body and its deformation by integrating the energy density function over the entire body Ω:

$$E[\phi] := \int_\Omega \Psi(F) d\vec{X} \tag{3.6}$$

However, that still gives us the sum of the strain energy in the body. We do not know the direction and location of the forces in the body. This prevents us from identifying deformations that achieve the necessary balance to satisfy the requirements for vibration modes. It also requires an extensive deformation mapping of the body. That is easy to get for simple shapes but mathematically intensive when our test objects get more complex. These problems can be solved by employing discretization.

3.2.1 Discretization

In simulations, the body is defined as the union of tetrahedra. We do not consider all possible deformations of the tetrahedra but restrict to deformations that are continuous and are given as an affine map in each tetrahedron. These deformations can then be mapped to the corresponding vertices of the tetrahedral mesh in space: $\vec{X}_1, \vec{X}_2, ..., \vec{X}_N$. We can define the deformation as the deformed vertex position, also known as degrees of freedom $x = (\vec{x}_1, \vec{x}_2, ..., \vec{x}_N)$ where $\vec{x}_i = \phi(\vec{X}_i)$. We can then define an energy function that takes the degrees of freedom as input, returning the total amount of strain energy.

$$E(x) := E[\phi] = \int_\Omega \Psi(F) d\vec{X} \tag{3.7}$$

[1] A mechanical property of the tension of a material `https://en.wikipedia.org/wiki/Young's_modulus`

[2] A mechanical property of the deformation of a material `https://en.wikipedia.org/wiki/Poisson's_ratio`

Figure 3.2: The first 10 eigenmodes of the armadillo mesh

Our function's input is now discrete, while our function requires a continuous answer. As mentioned before, it is possible to interpolate between the points and compute complex mathematical functions that would define the interpolation, but that is computationally expensive. It is, therefore, better to find a definition for the deformation map that is based on the vertices of our simulated body.

Simulating the body requires a digital model that allows easy volume calculation. We will be using a tetrahedral mesh. The total volume is then calculated by summing the volumes of each tetrahedron together. The tetrahedron is also a great shape for reconstructing the deformation map ϕ. It is possible to define an *affine map* for each tetrahedron T_i.

$$\phi(\vec{X}) = A_i\vec{X} + \vec{b}_i \text{ for all } \vec{X} \in T_i \tag{3.8}$$

3.3 Modal Analysis

Modal analysis is used when one is analyzing a spectrum of frequencies or related quantities. An extension of modal analysis is spectral theory. In spectral theory, one uses eigenvalues and eigenvectors as operators to analyse frequencies. We use this theory as the basis for finding eigenmodes

Eigenmodes, also known as vibration modes, are distinct patterns of vibration that a structure or object can exhibit. Every eigenmode corresponds to a specific frequency and respective shape of vibration.

Computing eigenmodes is done by solving for eigenpairs. In our case, we solve for eigenpairs in large sparse matrices. Solving for eigenpairs in a large sparse matrix requires methods specifically tailored for large sparse problems. Here, we will introduce the subspace iteration method, one of these methods.

3.4 The Subspace Iteration method

The subspace iteration method found in the article by Bathe and Wilson [1973] in 1972 aimed to solve for frequencies and mode shapes of structures with a particular focus on buildings and bridges. The subspace iteration method solves generalized eigenproblems of the form shown in equation (3.9), where S and M are the respective stiffness and mass matrices of size $n \times n$. Φ is equal to the set of eigenvectors of size $n \times p$, where p is equal to the amount of smallest eigenpairs to be calculated. Accompanied by Ω of size $p \times p$, a

diagonal matrix with the resulting eigenvalues on diagonal. The Subspace Iteration Method (SIM) will solve for the p smallest eigenvalues λ_i and their respective eigenvectors ϕ_i where $i = 1, 2, .., p$. The final result will be ordered in the form: $0 < \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_p$

$$S\Phi = M\Phi\Omega \tag{3.9}$$

SIM works by slowly iterating to the solution starting from the initial iteration vectors. The vectors are column entries in an n by q matrix Φ. Where q is larger than p. How much larger is often determined by testing multiple values. A term between $1.5p$ and $2p$ is considered normal. Iterations generally follow the pattern shown in the equations below: (for iteration $k = 1, 2, ...$)

After each iteration, the resulting iteration vectors are tested for convergence. In our test setup, we use the convergence test shown in Nasikun and Hildebrandt [2022]:

$$\frac{||S\Phi_i - \lambda_i M\Phi_i||_{M^{-1}}}{||S\Phi_i||_{M^{-1}}} < \varepsilon \tag{3.10}$$

Algorithm 1 Subspace Iteration Method from Nasikun and Hildebrandt [2022]

Input: Stiffness matrix $S \in \Re^{n \times n}$, mass matrix $M \in \Re^{n \times n}$, initial vectors $\Phi \in \Re^{n \times q}$, number of eigenpairs p, tolerance ε, shifting value μ

Output: Matrix Δ with lowest eigenvalues of (3.9) on diagonal and Φ listing eigenvectors as columns. First p pairs converged.

1: **function** SIM(S, M, Φ, p, ε, μ)
2: Compute sparse factorization: $LDL^T = S - \mu M$
3: **repeat**
4: Solve using factorization: $(S - \mu M)\Psi = M\Phi$
5: Compute reduced stiffness matrix $\bar{S} \leftarrow \Psi^T S \Psi$
6: Compute reduced mass matrix $\bar{M} \leftarrow \Psi^T M \Psi$
7: Solve dense eigenproblem: $\bar{S}\bar{\Phi} = \bar{M}\bar{\Phi}\bar{\Delta}$
8: Update vectors $\Phi \leftarrow \Psi\bar{\Phi}$
9: **until** pairs $(\bar{\Delta}_{ii}, \Phi_i)$ pass convergence test (3.10) for all $i \leq p$ **return:** $\bar{\Delta}$ and Φ
10: **end function**

SIM benefits from good initial input vectors. The distance of the input directly correlates to the number of iterations that are required to solve the eigenproblem within the specified tolerance. A lot of research on SIM puts the focus on improving the initial input. Bathe and Ramaswamy [1980a] tested initial vectors initialized with random content as well as initial vectors produced by the Lanczos process, Cheu et al. [1987b] investigated the effects of selecting initial vectors on computation speed and Wang and Zhou [1999] eventually proposed a best over-relaxation factor for each individual vector to be used for initialization.

Another optimization method for SIM is the use of shifting. Shifting can be applied to the stiffness matrix to shift the search space for the eigenpairs favourably. SIM solves for p eigenpairs where the first solved pair is equal to λ_1 and the last solved pair is equal to λ_p. Shifting the search space allows SIM to more quickly solve for the first pair. A method proposed by Zhao et al. [2007] shows the results of a good shifting strategy.

3.5 The Hierarchical Subspace Iteration method

The Hierarchical Subspace Iteration Method (HSIM) by Nasikun and Hildebrandt [2022] is another optimization method. HSIM creates multiple levels with the use of vertex sampling and a prolongation operator. It then solves the coarsest level densely, of which the output is then used as input for the next level where it uses SIM. It continues to do this until it solves the finest level with the output of the previous level as its input. This method takes advantage of the available context of the problem.

We will explain all the required steps for HSIM to work in more detail in this section since our work applies HSIM to our problem.

3.5.1 Hierarchy construction

We start off with constructing a vertex hierarchy to be used for our sampling. It is important that the hierarchy covers the mesh as well as uniformly as possible in order to promote overlapping regions. HSIM uses farthest point sampling as its sampling algorithm.

Sampling is, however, only part of the construction. Another important factor is the level of size. The size of the levels corresponds directly to the size of the finest level and the amount of levels. The levels T range from 0 to $T-1$. Where the coarsest level is defined in equation (3.11).

$$n^{T-1} = max[1.5p], 1000 \tag{3.11}$$

Nasikun and Hildebrandt [2022] define a growth rate μ (3.13) to use between levels.

$$n^T = \mu n^{T+1} \tag{3.12}$$

Where μ is given by the Tth order square root of the ratio of the finest level divided by the coarsest level.

$$\mu = \sqrt[T]{\frac{n^0}{n^{T-1}}} \tag{3.13}$$

3.5.2 Prolongation and restriction

Defining the levels is an important step, but not the only one. Next Nasikun and Hildebrandt [2022] define prolongation and restriction operators that allow us to restrict and scale the problems and the solution from one level to another. These operators are each other's transpose and define the influence of a sample on their neighbours.

To determine the factor of influence Nasikun and Hildebrandt [2022] use the geodesic distance of a vertex from the selected sample. They scale this distance linearly between 0 and the max distance p^T defined in equation (3.14).

$$p^T = \sqrt{\frac{\sigma A}{n^T \pi}} \tag{3.14}$$

Where A is the area of the surface and σ is a control parameter. The reasoning behind equation (3.14) is that the control parameter σ specifies the overlap between geodesic disks. Where σ stands for the amount of times the geodesic disks can cover the entire object

area. The vertices within the area of influence of a geodesic disk are then entered in the prolongation operator at their respective entry. The prolongation operator is then normalized row-wise to end up with our final prolongation operator. The stiffness- and mass matrices are then computed for each level by using the restriction operator.

3.6 Turning vectors

Our algorithm table of the Enriched Subspace Iteration Method proposed by Kim and Bathe [2017] can be seen in algorithm 2.

Algorithm 2 Enriched Subspace Iteration Method from Kim and Bathe [2017]

Input: Stiffness matrix $S \in \Re^{n\times n}$, mass matrix $M \in \Re^{n\times n}$, initial vectors $\Phi \in \Re^{n\times q}$, number of eigenpairs p, tolerance ε, shifting value μ

Output: Matrix Δ with lowest eigenvalues of (3.9) on diagonal and Φ listing eigenvectors as columns. First p pairs converged.

1: **function** ENRICHED SIM(S, M, Φ, p, ε, μ)
2: Compute sparse factorization: $LDL^T = S - \mu M$
3: Initialize orthonormal starting iteration vectors with 1 iteration $SIM(S, M, \Phi, p, \varepsilon, \mu)$
4: **repeat**
5: Partition iteration vectors $\Phi_k = [\Phi_{pk}, \Psi_k^a, \Psi_k^b]$ ▷ Φ_{pk}: converged vectors
6: Solve with factorization $S\bar{\Psi}_{k+1}^a = M\Psi_k^a$
7: Construct Y_k according to Kim and Bathe [2017](2.1.3)
8: Solve with factorization $S\bar{Y}_{k+1}^a = MY_k^a$
9: Construct $\bar{\Psi}_{k+1} = [\Phi_p k, \bar{\Psi}_{k+1}^a, \bar{Y}_{k+1}]$
10: Compute reduced stiffness matrix $\bar{S}_{k+1} \leftarrow \bar{\Psi}_{k+1}^T S\bar{\Psi}_{k+1}$
11: Compute reduced mass matrix $\bar{M}_{k+1} \leftarrow \bar{\Psi}_{k+1}^T M\bar{\Psi}_{k+1}$
12: Solve dense eigenproblem: $\bar{S}_{k+1}\bar{\Phi}_{k+1} = \bar{M}_{k+1}\bar{\Phi}_{k+1}\bar{\Delta}_{k+1}$
13: Update vectors $\Phi_{k+1} \leftarrow \bar{\Psi}_{k+1}\bar{\Phi}_{k+1}$
14: **until** pairs $(\bar{\Delta}_{ii}, \Phi_i)$ pass convergence test (3.10) for all $i \leq p$ **return:** $\bar{\Delta}$ and Φ
15: **end function**

The steps of the Enriched subspace iteration method are similar to those of the original subspace iteration method. The difference is the addition of steps 5-9. We will give a brief description of all the steps in the algorithm. Step 2 computes the sparse factorization for the shifted stiffness matrix. This factorization can then be used by both Enriched SIM and the initial SIM call. In step 3, the original SIM algorithm described in 1 will be run for 1 iteration. This creates a set of orthonormal starting iteration vectors for enhanced SIM. In steps 4-14, we repeatedly execute enriched sim iterations until. In step 5, the iteration vectors are split into 3 partitions. The first partition $\Phi_p k$ is equal to the subset of p_k converged vectors. The other two sets are order n x r_k where $r_k = (q - p_k)/2$. This gives us two equal sets of unconverted vectors. In step 6, we solve the linear system for all vectors in Φ_k^a. The results are then used to generate and apply turning vectors in step 7. A further explanation of the math is provided in Kim and Bathe [2017](2.1.3). The turned vectors

are then used to solve the linear system. After which, we construct our set of iteration vectors $\bar{\Psi}_{k+1}$. We then start a normal SIM iteration in steps 10-13, after which we check for convergence and repeat the iteration if not all p eigenpairs have converged.

Chapter 4

Method

In this chapter, we discuss the approach and the respective considerations for the research questions. We first describe the basic changes that had to be made to fit HSIM onto our problem. We then discuss changes in parameters which will better fit the algorithm to our problem, as well as introduce a few novel ideas that aid the speed of SIM iterations for our problem.

4.1 Baseline Hierarchy Performance

Since our goal is to adapt HSIM to the elasticity problem, it is important to establish a baseline performance reading comparing HSIM to SIM. However, before we can evaluate the baseline performance of HSIM on our problem, we have to adapt HSIM to work with our problem specification. Adapting HSIM to work requires multiple changes. Firstly, we have to adapt working metrics. HSIM uses the total surface area as a parameter for the size of the radius of influence used in formula (3.14). For our adaptation, we will continue to use the surface area, which requires us to supply the surface triangles for the meshes that we do not tetrahedralize ourselves. Another necessary adaption is the degrees of freedom per sample as well as general indexing within the hierarchy. Every vertex represents 3 degrees of freedom now, which requires extensive changes to code to accommodate for that. We decided to continue sampling per vertex instead of per degree of freedom. Apart from these changes, we implemented the hessian matrix generation as well as an extensive clamping library to prevent eigenvalues equal to 0. This allowed us to collect a baseline performance reading.

The baseline can be established by using the parameters defined in the original HSIM paper. We do have to adapt the algorithm to accept meshes where each vertex corresponds to 3 degrees of freedom. After that, we disable all the SIM-specific optimizations: double solve step, excluding converged iteration vectors and shifting. Comparing HSIM and SIM without any other optimizations except for the hierarchy for HSIM allows for a good performance indication that the hierarchy brings.

This experiment for testing the hierarchy performance might give negative results or even not converge at all if the hierarchy returns linearly dependent iteration vectors. The

experiment is meant to serve as a starting point for experimentation and parameter determination. Furthermore, it will aid as a point of comparison for additional optimization or adaptations to the method. We might encounter scenarios where parameters improve performance when only certain conditions are met. In that case, we can compare the overall performance to the results of this experiment.

Our results will include the iteration count of both SIM and HSIM, as well as their respective real-world timings. The timings of HSIM will be split out between hierarchy timing and the total solve time. The results will be accompanied by an analysis of the relative difference between both algorithms produced within their specified tolerance. A plot of the Fourier coefficients of the eigenfunctions of the coarse level in the eigenbasis of the finest level will be included as well. Plotting the Fourier coefficients, seen in a paper by Nasikun et al. [2018] in Figure 7, gives a good insight into how well the solution of the coarser levels translates to the finest level.

4.2 Basis Functions

The construction of the prolongation and restriction operators depends on multiple parameters and design decisions. In this section, we will focus on a few of those parameters and design decisions. We will explain why they are important as well as hypothesise why the optimal value or strategy might be different for our application of HSIM compared to the original application of the Laplace-Beltrami problem.

4.2.1 Support Region

When constructing the basis, HSIM makes use of a radius of influence for each sample. This radius is used to translate the influence of the selected vertex onto all vertices within the radius. The length of the radius differs per level and is computed with the equation shown in 3.14. The variable σ was determined on 7 in the HSIM paper through heuristics. The authors theorize that this value agrees with the average valence in a triangle mesh. The value should be set to the average expected number of non-zero entries per row in the prolongation operator.

Our problem requires constructing a hierarchy of a volumetric mesh instead of a surface mesh. This comes with a higher expected average valence. Another possible factor at play here can be the amount of degrees of freedom for each vertex. This results in 3 entries per vertex in the prolongation operator instead of 1 entry per vertex for the Laplace-Beltrami operator.

Exploring the effects of different support values on our problem might result in a better value tailored to our problem. We can determine this value by heuristics. In our implementation, we enter 3 entries per vertex instead of 1. Since this step happens after the max distance determination, it does not impact the support value. The average valence, however, still does.

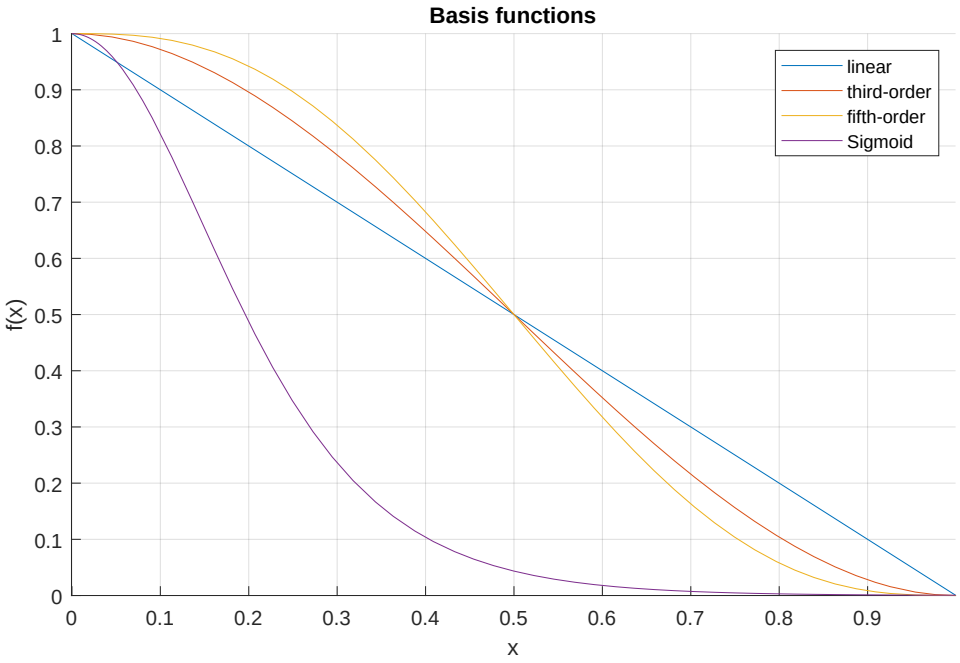


Figure 4.1: A graph showing the different basis functions we will test for basis construction.

4.2.2 Functions for basis construction

The prolongation and restriction operators of the hierarchy are constructed by mapping the influence of the sampled vertices of the lower level to the vertices at the higher level. While the degrees of freedom per vertex differ, in our case, the method remains the same. To determine the influence of one vertex on a sampled vertex, we use a mapping function that maps Dijkstra's distance between a value of 0 to 1, depending on the maximal distance.

HSIM makes use of linear mapping. There is, however, a previous paper from the authors Nasikun et al. [2018] that makes use of a third-order polynomial to map influence. The authors discuss a selection of basis functions in a supplementary document Nasikun et al.. They discuss a few other basis functions. We will test these basis functions for our application together with an optional Gaussian function. This function is more aggressive in its drop in influence.

Our testing methodology will differ from the methodology of the supplementary document by Nasikun et al.. The document shows testing done with Laplace-Beltrami problems while we aim to use it for the elasticity problem. The fast approximation paper aims to achieve good approximations by immediately prolongation the coarse level. Our aim differs partially. We aim for a good translation between levels in order to improve convergence speed on the finest level.

4.2.3 Euclidean Distance

HSIM uses an approximation of the geodesic distance between vertices by using Dijkstra's algorithm. This distance calculation is used for farthest point sampling when creating the vertex hierarchy. It is also used for the basis construction, where the influence of a vertex on another is defined by the distance between them. It would be interesting to see if the usage of Euclidean distance improves the basis construction. The geodesic distance works well on surface meshes since it allows for a large sphere of influence without including vertices that have no direct connection to the selected vertex within the sphere of influence. However, we are working with volumetric meshes. That should mostly prevent these far-away yet included vertices when using the Euclidean distance, except in certain cases.

We can test this hypothesis by testing the Euclidean distance implementation against the original implementation.

Using the Euclidean distance might have little impact on the performance of the algorithm. If that is the case, it might be a logical option to explore this problem since it would allow for a cheaper hierarchy construction. For these tests, we will look at the performance of the algorithm by comparing the number of iterations, solve time and hierarchy construction time.

Another possible benefit of using the Euclidean distance is the possibility of parallel hierarchy generation. This could prevent bottlenecks for high vertex count models. It would also provide a simpler implementation path where one could quickly construct hierarchies with little to no abstractions. We combine the availability of Euclidean distance with a multitude of distance functions that allow us to map the influence of a vertex to another in a non-linear way.

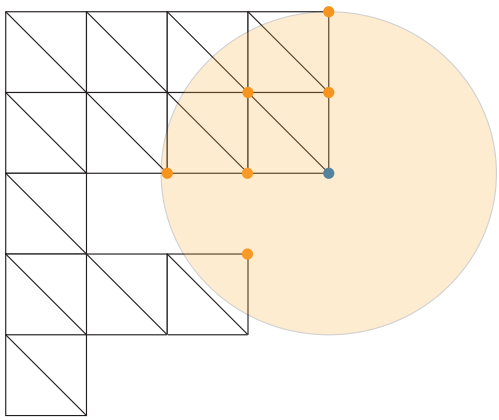


Figure 4.2: An 2D example showing the air-gapping that can occur when using Euclidean distance. The vertex on the bottom of the mesh has no path to the sampled vertex within the circle of influence.

4.3 Degrees of Freedom per Level

The size of the submatrices for each level is decided by equation 3.12. This equation bases the size of the intermediate levels on the size of the coarsest and finest levels. The coarsest level size is user-defined. In the original HSIM paper, they set this level at $\max(1.5p, 1000)$. Our coarse level size should allow efficient dense solving while allowing for a good solution that performs well on iteration vectors on the level above.

Our implementation maintains the sampling per vertex methodology of the original HSIM implementation. The only change we make is that we now include three degrees of freedom per sampled vertex instead of only one. Another possible method of sampling is discussed in Section 6.3.3. The method discussed in that Section is not applicable to our current focus on isotropic materials.

To test for the optimal amount of degrees of freedom, we will look at the performance of the coarse level solve together with the effects on the number of iterations on the finest level. We will also graph the norm of the difference of the prolongated coarse level results projected onto the reference solution of the finest level. This plot will show the effects of coarse level size on the correctness of the prolonged solution. Our goal is to find a proper size that gives the lowest difference while allowing for efficient dense solving.

4.3.1 Number of levels

The optimal number of levels used for solving depends on a few factors. Every level requires a new matrix factorization, which is computationally expensive. The benefit, however, is a possible reduction in iterations for the finest level. Computation steps on the finest level are more expensive due to the size of the matrices. If an extra level is capable of reducing the compute time on the finest level by a larger amount than the required compute time for the extra level, then we can consider it a worthy trade-off. The original HSIM paper found that for their application, there were no benefits when the number of levels grew beyond 3. They also found that 2 levels were enough for a small number of eigenvalues $p < 200$, and 3 were better for larger numbers. The tests will be run on 2 different objects differing in vertex count. The authors of HSIM, Nasikun and Hildebrandt [2022], tested different tolerances as well but found that the optimal number of levels does not change for different tolerances. We decided to only test with the tolerance $1e-2$ since we expect HSIM to perform best for large tolerances.

To test the performance of different numbers of levels, we can create a table of different runs with varying numbers of levels ranging from 2 to 4. In this table, we can list the run-time and iterations used per level. It will also be useful to include the maximum norm difference of the solution of every sub-level to the reference solution. We should use the first p vectors for this comparison since we only converge the first p eigenpairs on each level. A secondary visualisation can be used to compare the variance in norm difference per level for 4 levels against 2. The timings for the factorization and solve time per level will also be put in a stacked bar chart where we can visualize the execution timings vertically.

4.4 Shifting

Subtracting the mass matrix times a shift value μ of the stiffness matrix before factorization is called shifting. The shift accelerates convergence near eigenvalues close to the shift value μ that is being used. The shift value is intended to be used multiple times during solving, where it would be the average of the last converged eigenvalue and the next eigenvalue to converge.

There is, however, another method that proposed an aggressive shift into the unconverged eigenvalue range by Zhao et al. [2007]. The shift value is set to an unconverged eigenvalue at index α where alpha is user-configurable. The original authors of the shifting strategy settled on $\alpha = 0.4$. The recommended value in the HSIM paper is $\alpha = 1/3$. HSIM makes use of this value in a different manner. The shift is only applied once per level, and the value μ is calculated by using the resulting eigenvalues of the previous level. The eigenvalues should be approximately the same between levels, which allows for a good shifting selection.

An optimal shifting value can be found by benchmarking the number of iterations and timings of a test problem for different numbers of eigenvalues. We can then highlight the best timing for each parameter per test run as well as the greatest reduction in iterations at the finest level. These should correspond since the shift parameter does not affect the computation speed of the factorization by much. We can then select the optimal shift value for our problem by choosing the value that guarantees the best overall performance.

4.5 Subspace Size

SIM makes use of a subspace to project the stiffness and mass matrices. The projected matrices are then densely solved, and the results are then lifted up from the subspace and used as input for the next generation. Increasing the subspace size can reduce the required number of iterations, but it also makes the iterations more costly. Making the subspace too small could slow down convergence significantly. It is, therefore, important to choose an appropriate subspace size that provides a good balance.

However, it would also be interesting to see if we can find a subspace size where we are able to reduce the number of required iterations down to 1. Exploring the effects of different subspace sizes while using the hierarchy might result in a strong enough reduction of iterations to justify the additional computational cost per iteration.

To test the effects of subspace size, we will measure the timings and number of iterations for differing subspace sizes and differing number of eigenvalues. Our parameters will be set to the resulting values from the previous experiments. We can support possible values by comparing the Fourier coefficients of the last eigenvector ϕ_p in the finest level on the prolongated solution of the coarse level. We can take the absolute of these of these coefficients and sum them from start to i, where i is an index of the subspace. Plotting these values results in a plot that converges to 0 at the index of the subspace where all the coefficients necessary for convergence are found 4.1.

$$1 - \sqrt{\sum_{i=0}^{j} a_j^2} \text{ for all } j \in 0, \ldots, q \tag{4.1}$$

The increase in subspace size does require the usage of additional memory for storing all the vectors. A combination of many requested eigenpairs with a large subspace size and a large model might result in system limitations. In those cases, we might leave out that specific entry while indicating that we were ¡memory bound¿ for that specific test. The model selection for the tests will be specified in the experiments where the initial experiments might indicate the possibility of including a larger model.

4.6 Tolerance

Another parameter that should be tested is the specified tolerance of the desired result. The authors of the HSIM paper specify a tolerance of 1e-2 for their problem and show the results of running HSIM with a tolerance of 1e-4. The results show better scaling for Lanczos' method compared to the SIM-based methods. We should test the effects of increasing tolerance for our problem as well.

We can make the hypothesis that the results will be equal to the ones shown in the HSIM paper. Computing a smaller tolerance would require more SIM iterations for both SIM and HSIM to satisfy the tolerance requirements for each eigenpair. It is, however, interesting to compare the results with Matlab Lanczos' algorithm as well.

Our results can be shown in a small table that compares HSIM's and SIM's performance with each other for a tolerance of 1e-2 and a tolerance of 1e-4.

4.7 Materials

Material parameters are used to build stiffness and mass matrices that reflect the materials of the real world with the same material parameters. These material parameters can change the condition of the matrices by increasing the number ranges or distributing energies more broadly within each tetrahedron. It is, therefore, important to test a few materials and their possible impact.

For our testing, we can test the relative performance of both rubber and steel for the material parameters in terms of timing. An interesting addition can be a graph where we change the parameters in an unrealistic manner where we list the iteration counts for both. We can use the values $v = \varepsilon, 0.10, 0.20, 0.30, 0.40, 0.50 - \varepsilon$ as Poisson's value. We can use the values $k = 0.01, 0.1, 1, 10, 100, 1000$ in GPa for Young's modulus.

4.8 Turning Vectors

In this section, we will discuss our considerations concerning the acceleration method of turning vectors by Kim and Bathe [2017]. Turning vectors aims to improve the quality of the iteration vectors during every iteration by turning the iteration vectors in a way that aids

faster convergence. This method showed a reduction of iterations and solve time when introduced, however, it was run in a sequential manner. This does not invalidate the reduction in iterations, but it might result in a slowdown compared to SIM when both are run with multiple cores available.

The algorithm requires extra sequential steps during every iteration, dependent on the amount of turning that is required for that iteration. The benefit of this step is the reduction in iterations. Our implementation applies parallel processing to the algorithm where it seemed possible, but we were unable to make it fully parallel. The algorithm also introduces the requirement to first run a normal SIM iteration before starting the enhanced SIM iterations to prevent iteration vectors from becoming linearly dependent when turning.

These extra requirements create conflicts with other optimizations that HSIM uses, such as the double-solve step per iteration, as well as the optimization where we do not solve for already converged vectors within a stricter tolerance. Enhanced SIM does ignore the already converged vectors as well, but it does not make use of a sort-like multi-solve step.

4.8.1 With Hierarchy

We think it is important to test the performance of enhanced SIM within the hierarchy. It can result in a reduction in iterations or show that the method does not contribute much to the hierarchy. It is possible to look at the total number of turns that the algorithm uses for a run. This can give an indication of its use. A low number of turning vectors indicates that the iteration vectors are already near the optimal state.

The requirement of a normal SIM step before it uses the turning vectors does limit its applications. We should compare the use of enhanced SIM for both the finest level and the intermediate levels. We can evaluate performance through known means such as timings and iterations, but we should also list the total amount of turning vectors used during every level.

Enhanced SIM introduces a new algorithm to test. We will compare the performance of HSIM against three different implementations of HSIM with turning vectors:

1. HSIM with turning vectors on the finest level
2. HSIM with turning vectors on all levels
3. HSIM with turning vectors on intermediate levels

Our motivation for the first and second implementations stems from our considerations above. We think it might improve the finest level convergence but hinder intermediate levels in iteration count. To test this hypothesis, we should, therefore, test both implementations 1 and 2. The last implementation stems from a hypothesis that can partially be proven by the second implementation. Since SIM with turning optimizes all the iteration vectors with turning. We think that it might be possible to introduce this optimization on the iteration vectors on an intermediate level and then prolong them to the finest level. This would significantly reduce the performance penalties of the turning vectors while possibly keeping the speed up in convergence.

4.8.2 Against Hierarchy

It is important to test enhanced SIM as a stand-alone SIM implementation as well. The results of the previous experiment in section 4.8.1 might indicate that the hierarchy and the turning vectors compete for subspace optimization. If that is the case, we should consider them as competitors during our final experiment. Getting a baseline reading to see the performance of enhanced SIM compared to HSIM and normal SIM is, therefore, an important insight.

We can make use of the existing implementation and run enhanced SIM with the same parameters as our SIM runs. We can then note down timings for both as well as the total amount of turning vectors used for enhanced SIM. We can present the result in a table and discuss whether or not to include enhanced SIM in our benchmarks as a competing algorithm.

4.9 Scaling

One of SIM's advantages is scale-ability. It is capable of using all available threads scaling appropriately, as shown by Bathe and Ramaswamy [1980b]. The algorithm also has an advantage when the number of searched eigenvalues increases. This stems from the use of the subspace. Every iteration improves the entire subspace. When the subspace is bigger due to a higher number of requested eigenvalues, it does not directly result in more iterations. Instead, the iterations can become more expensive.

4.9.1 Number of EigenValues

HSIM for Laplace-Beltrami problems shows strong scaling improvements when focusing on the number of eigenvalues when comparing it to the original SIM algorithm. It is important to verify if this scaling is also present in our application. It will provide insight into the applicability of HSIM for vibration modes and perhaps find use cases where it is always more beneficial to use HSIM for a certain number of eigenvalues and up. We can test the absolute and relative performance increase of HSIM, SIM and Lanczos based on the number of eigenvalues to compute by testing at least 2 different models where the amount of requested eigenvalues is varied. The results will include a table where the timings and iteration counts are listed for both SIM and HSIM, with Lanczos only having timings listed. A graph of the timings per algorithm will also be included to visualize the scaling.

4.9.2 Parallel Scaling

For these tests, we will compare the performance per number of cores of HSIM and SIM in order to determine HSIM relative scaling when the number of cores is increased and to compare it to SIM. The original SIM algorithm scales well when increasing the number of cores, as shown in Bathe and Ramaswamy [1980b]. It is important to establish if this holds up for HSIM as well. And how well it compares to SIM. The test gives guidance as to when one should prefer one over the other. If HSIM scales better than SIM, one might prefer

running HSIM on a cluster, but if it seems to have a worse scaling factor than SIM, there might be a certain number of cores that define the border condition on choosing SIM over HSIM.

To test this, we will disable the cores within the bios itself. The tests will still be run on Windows 11, where an operating system is present. This might result in strange performance numbers when testing low core counts. The experimentation will show if the data is comparable to the graphs shown in Bathe and Ramaswamy [1980b]. It might be possible that the OS will increase its background usage when the number of cores is increased again.

To ease testing time, we can make use of a small model with a small amount of degrees of freedom. This would allow us to test for a small amount of eigenvalues as well as a somewhat bigger amount to compare the scaling factors of both. Testing two different numbers of eigenvalues will allow us to see if the increase in parallel performance is dependent on the amount of the eigenvalues and to what extent. Our hypothesis would be that an increase in the number of eigenvalues increases the parallel scaling factor of the algorithm. Since most parallel steps execute on a number of subspace columns often equal to $q/core\_count$.

The results will include a table with timings for SIM and HSIM accompanied by a plot containing the interpolated time measurements per algorithm per core count. The computer that will be used for testing has an 8-core processor available. Which allows for 8 data points when excluding multi-threading.

Chapter 5

Experimentation

5.1 Experiment setup

In this chapter, we specify our experimentation setup and clarify implementation details that might impact the experiments in one way or another. All our experiments are run on an 8-core Ryzen 7 3700X with 64GB of RAM combined with an RTX 2060 super with 8GB VRAM. The implementation makes use of Eigen [Guennebaud et al., 2010] for linear algebra computation combined with LibIGL [Jacobson and Panozzo, 2017] for its geometric implementations and integration with tetGen [Si, 2015]. The implementation includes integrating with Intel's Pardiso solver [Schenk and Gärtner, 2011]. The parallel computations are done with OpenMP.

5.2 Test models and problem generation

For our testing setup, we decided to generate our volumetric models and their respective stiffness and mass matrices during runtime or by retrieving them from the cache. This allows us to modify the material parameters of a model, and it allows us to test a wide range of publicly available models. Most of our experiments are run with the same material parameters unless specified otherwise. The default parameters are: $k = 200GPa$, $v = 0.27$, density $= 7750kg/m^3$ which are the parameters used for steel.

5.2.1 Problem generation

It is also important to establish a workflow for problem generation. Our application requires not only the stiffness- and mass matrices but also the associated mesh. Finding tetrahedralized meshes with the correct stiffness- and mass matrices for the linear elasticity structure is hard, and creating a proper testing set out of the rare finds without being able to validate the correctness of the matrices would yield unusable results. Therefore, we decided to implement the problem generation as well as the tetrahedralization of surface models without boundaries ourselves.

5.2.2 Test models

Most extensive testing has been done with two models: The Rocker arm model and the Armadillo model. Other models are used in the experiments as well. All used models can be seen in Figure 5.1. For a complete list of the models, including the vertex count and sources, we refer to Appendix B.

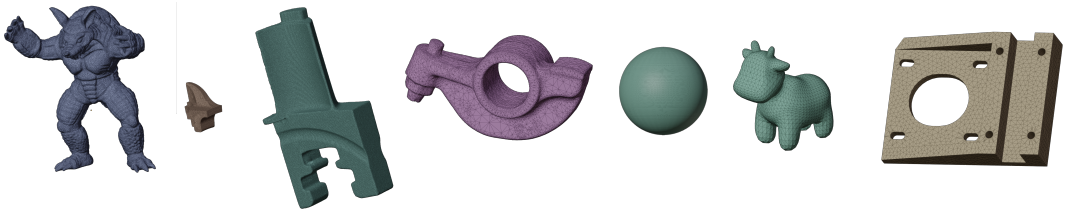


Figure 5.1: All the models used for testing. Naming (left to right): Armadillo, Fandisk, Blade smooth, Rocker arm, Sphere, Spot, Stepper body

5.2.3 Measurements

Timings are measured using a singleton class using the std::chrono library to calculate the duration between two measurement points in microseconds. All experiments were run on a Windows 11 with all non-windows applications and background processes killed. The resulting measurements were appended to a CSV file together with the given parameters for the calculation.

The source code is compiled with Intel® oneAPI DPC++/C++ Compiler version 2023.2 as release distribution with the following flags:

```
-march=core-avx2 -O3 -qopenmp -W -Wall -pedantic -fma
```

The benchmarks were run with a Windows Batch file that would call the executable with the parameter flags for every run. Starting the executable for every run terminates the possibility of run time differences for sequential runs. Our code makes use of Open MP thread pools and extensive amounts of memory. Both are hard to reset within our code since they also depend on the resource allocation of the operating system. Starting a new process for every run ensures that our code can not make any claims on still-existing resources.

This method has not been applied to Matlab's Lanczos solver. There, we make use of loops to run the algorithm together with tic and toc. The algorithm will be run 5 times, and the average will be taken. During our exploratory research, we saw no direct improvement or decrease in performance when consecutively running the same function.

5.2.4 Notation

Our experiments show the number of iterations required for HSIM to complete in tables. Our notation for the number of iterations follows the notation shown in the original HSIM paper [Nasikun and Hildebrandt, 2022]. The paper uses the letter F to denote the coarse

level computation followed by the number of iterations per level separated with |. An example of a 3-level computation where the finest level took 5 iterations and the intermediate level took 3 iterations would be written like this: $F|3|5$.

5.3 Baseline Hierarchy Performance

In this experiment, we test the performance of HSIM, SIM and Matlab Lanczos' solver. The parameters for HSIM are set equal to the recommended parameters specified in the HSIM paper. We disabled the shifting for SIM since our implementation's shift depends on the eigenvalues of the previous coarser level. Table 5.1 shows that HSIM is faster than SIM

Object	Eigvals	HSIM			SIM		Matlab
		Hierarchy	Its	Total solve time	Its	Solve time	Solve time
Rockerarm (∼56k DoF)	50	0.8	$F\|3$	13.0	9	18.7	2.3
	250	1.5	$F\|3\|4$	38.4	11	55.7	12.1
	500	1.4	$F\|4\|6$	72.7	12	103.3	41.2
Armadillo (∼693k DoF)	50	12.6	$F\|3$	86.2	8	367.2	41.0
	250	17.8	$F\|4\|5$	473.8	13	1366.1	200.9
	500	17.1	$F\|11\|6$	1057.1	14	2868.6	500.9

Table 5.1: A table showing the performance of HSIM, SIM and Matlab on the test system time in seconds

for runs in the table. HSIM has half the amount of required iterations on the finest level compared to the iterations for SIM. HSIM is, however, still slower than the Lanczos solver from Matlab.

5.4 Basis construction

5.4.1 Support Region

In this experiment, we test the relative performance of HSIM for different support region values σ as seen in Equation (3.14). However, during our testing, we configured another distance function than originally planned. We decided to re-run the benchmark with the correct distance function while also including the results of the other distance function.

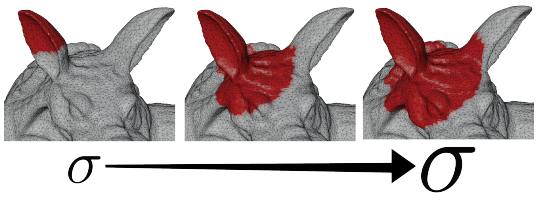


Figure 5.2: A Figure showing the effect of increasing the support region value on the vertices within the radius of influence. A larger number results in a larger radius of influence.

Linear basis function	Support	$\sigma = 5$		$\sigma = 7$		$\sigma = 9$		$\sigma = 11$		$\sigma = 13$		$\sigma = 15$	
Object	Eigvals	its	time	its	time	its	time	its	time	its	time	its	time
Rockerarm (∼56k DoF)	50	$F\|4$	10.0	$F\|3$	8.8	$F\|3$	8.9	$F\|3$	9.2	$F\|3$	9.5	$F\|3$	9.8
	250	$F\|3\|5$	32.3	$F\|3\|5$	35.1	$F\|3\|4$	35.1	$F\|3\|4$	37.9	$F\|3\|4$	41.0	$F\|3\|4$	43.9
	500	$F\|4\|6$	60.3	$F\|4\|6$	66.3	$F\|4\|5$	69.3	$F\|4\|5$	75.2	$F\|4\|5$	83.5	$F\|4\|5$	89.1
Armadillo (∼693k DoF)	50	$F\|3$	76.0	$F\|3$	76.9	$F\|3$	79.3	$F\|3$	83.1	$F\|3$	87.4	$F\|3$	94.9
	250	$F\|4\|5$	405.1	$F\|5\|5$	450.6	$F\|5\|4$	375.0	$F\|5\|4$	396.0	$F\|5\|4$	416.6	$F\|5\|4$	433.4

Table 5.2: Comparison of required iterations and solving time for different support parameters using a linear function as a distance function.

Cubic polynomial basis function	Support	$\sigma = 5$		$\sigma = 7$		$\sigma = 9$		$\sigma = 11$		$\sigma = 13$		$\sigma = 15$	
Object	Eigvals	its	time	its	time	its	time	its	time	its	time	its	time
Rockerarm (∼56k DoF)	50	$F\|4$	9.9	$F\|3$	8.6	$F\|3$	8.8	$F\|3$	9.0	$F\|3$	9.2	$F\|3$	9.6
	250	$F\|3\|5$	32.4	$F\|3\|4$	31.9	$F\|3\|4$	34.4	$F\|3\|4$	37.1	$F\|3\|4$	40.2	$F\|4\|4$	48.2
	500	$F\|5\|6$	64.7	$F\|4\|6$	65.7	$F\|4\|5$	68.8	$F\|4\|5$	75.2	$F\|4\|5$	81.2	$F\|4\|5$	88.0
Armadillo (∼693k DoF)	50	$F\|3$	74.7	$F\|3$	76.7	$F\|2$	64.8	$F\|3$	84.6	$F\|3$	88.2	$F\|3$	93.1
	250	$F\|4\|6$	465.5	$F\|4\|5$	428.7	$F\|5\|4$	377.8	$F\|5\|4$	397.6	$F\|5\|4$	417.4	$F\|5\|4$	438.3

Table 5.3: Comparison of required iterations and solve time for different support parameters using the cubic polynomial as a distance function.

This section includes two tables where table 5.2 compares the impact of different support region values while using linear distance mapping for the sphere of influence when constructing the prolongation operator. Table 5.3 compares the impact of different support region values while using the cubic polynomial from Nasikun et al. as a mapping function. Table 5.2 shows a decrease in required iterations on the finest level when $\sigma = 9$ and the number of eigenpairs to calculate is higher than 50. However, the rocker arm problem converged faster for lower values of σ.

Table 5.3 shows a decrease in the number of required iterations on the finest level for $\sigma = 9$ for the armadillo as well as the rockerarm when calculating 500 eigenpairs. A reduction in timings is still strongly present in the rockerarm runs where $\sigma = 5|7$. The Armadillo runs do show a significant time reduction for $\sigma = 9$.

An interesting observation can be made when looking at the results of the support parameter benchmark in Table 5.3. Where we can see the effects of the support value differ for both meshes. The Armadillo object benefits from a support value of $\sigma = 9$, reducing the number of required iterations on the finest level. The same effect is present for the rocker arm object where a reduction in the number of required iterations on the finest level is present for all runs at $\sigma = 9$. However, the optimal timings are present for lower support values. This can be attributed to lower hierarchy construction costs associated with decreasing the support value. The same effect can be seen in Table 4 in [Nasikun and Hildebrandt, 2022].

5.4.2 Functions for basis construction

		cubic polynomial			Linear polynomial		
Object	Eigvals	Hierarchy	Its	Total solve time	Hierarchy	Its	Total solve time
Rockerarm (∼56k DoF)	50	0.9	$F\|3$	8.8	0.9	$F\|3$	9.1
	250	0.9	$F\|3\|4$	34.4	1.7	$F\|3\|4$	34.9
	500	1.6	$F\|4\|5$	68.8	1.5	$F\|4\|5$	69.7
	1000	1.6	$F\|6\|9$	195.4	1.8	$F\|6\|8$	190.0
Armadillo (∼693k DoF)	50	14.3	$F\|2$	64.8	14.9	$F\|3$	81.8
	250	18.3	$F\|5\|4$	377.8	19.2	$F\|5\|4$	385.9

Table 5.4: A table showing the performance difference between the cubic polynomial and the linear polynomial. The Gaussian and Sigmoid basis functions are omitted since they introduced instability.

In this section, we compare the performance of various basis functions for HSIM as discussed in section 4.2.2. The basis function is used to map the influence of a vertex within the radius of influence of the sampled vertex between a value of 0 to 1 based on the distance of the sampled vertex. The differences in timing and iterations for the Rocker arm are small. The differences for the armadillo mesh are larger. The cubic polynomial reduces the number of required iterations on the finest level for the 50 eigenpairs run compared to the linear function. It also improves the total solve time for both runs. The results show that the hierarchy construction benefits from using the cubic polynomial as the basis function instead of the Linear function used in HSIM for Laplace-Beltrami problems.

5.4.3 Euclidean distance

		Geodesic distance			Euclidean		
Object	Eigvals	Hierarchy	Its	total solve time	Hierarchy	Its	Total solve time
Rockerarm (∼56k DoF)	50	0.9	$F\|3$	8.8	1.1	$F\|3$	9.1
	250	0.9	$F\|3\|4$	34.4	2.0	$F\|4\|4$	40.7
	500	1.6	$F\|4\|5$	68.8	2.1	$F\|5\|5$	78.8
Armadillo (∼693k DoF)	50	14.3	$F\|2$	64.8	16.5	$F\|3$	81.3
	250	18.3	$F\|5\|4$	377.8	20.2	$F\|6\|4$	388.5
	500	18.9	$F\|4\|5$	831.0	20.1	$F\|5\|5$	862.3

Table 5.5: A table shows the hierarchy construction time (seconds), the amount of required iterations, and the total solve time (seconds).

In this section, we compare the performance of using the geodesic distance compared to using the Euclidean distance. We compare hierarchy construction time, iteration counts and the total solve time. Table 5.5 shows that the Euclidean distance requires more time to construct the hierarchy while also increasing the total solve time of the algorithm for all runs. The runs with more than two levels show an increase in the number of iterations for the intermediate levels when using the Euclidean distance.

The use of Euclidean distance instead of geodesic distance has shown to be hurting performance in general. Our implementation for Euclidean distance use was not as opti-

mized as the approximated geodesic distance implementation, which hurts the hierarchy construction. However, comparing the differences in convergence time while disregarding the hierarchy construction time allows us to see that the Euclidean distance hurts the speed of convergence. This can also be seen in the iteration counts for all but the first row in table 5.5.

5.5 Degrees of freedom per Level

DoF on coarsest level		1500		3000		4500		6000		7500		9000	
Object	Eigvals	Its	Time	Its	Time	Its	Time	Its	Time	Its	Time	Its	Time
Rockerarm (∼56k DoF)	50	F|4	9.6	F|3	8.7	F|3	11.4	F|3	14.9	F|3	21.2	F|3	34.2
	500	F|5|6	66.2	F|4|5	66.5	F|4|5	74.8	F|4|5	82.3	F|4|5	92.6	F|4|5	109.8
Armadillo (∼693k DoF)	50	F|3	81.3	F|3	80.3	F|2	95.1	F|3	84.6	F|3	90.8	F|3	102.6
	250	F|4|4	356.1	F|4|4	367.5	F|4|4	385.3	F|5|4	410.9	F|4|5	463.4	F|4|4	437.4

Table 5.6: A table showing the performance of HSIM when changing the minimum coarse level size. Solve time is in seconds.

The number of degrees of freedom per level is decided by the function 3.12. The size of intermediate levels is based on the coarse level size. The coarse level size in our test setup is limited by our GPU's video RAM. The results in table 5.6 show overall higher timings for coarse level sizes with 4500 degrees of freedom or more. Increasing the degrees of freedom on the coarsest level results in a decrease in the number of required iterations, but the runtime suffers from the additional computational costs of solving the coarsest level. This slowdown is mainly caused by the construction of the mass- and stiffness matrices for the coarser levels.

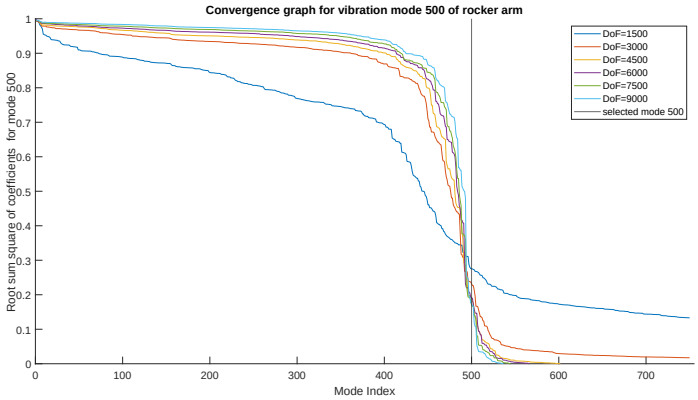


Figure 5.3: A graph showing the root sum square of the Fourier coefficients for the 500Th mode for varying coarse level sizes for the rocker arm model

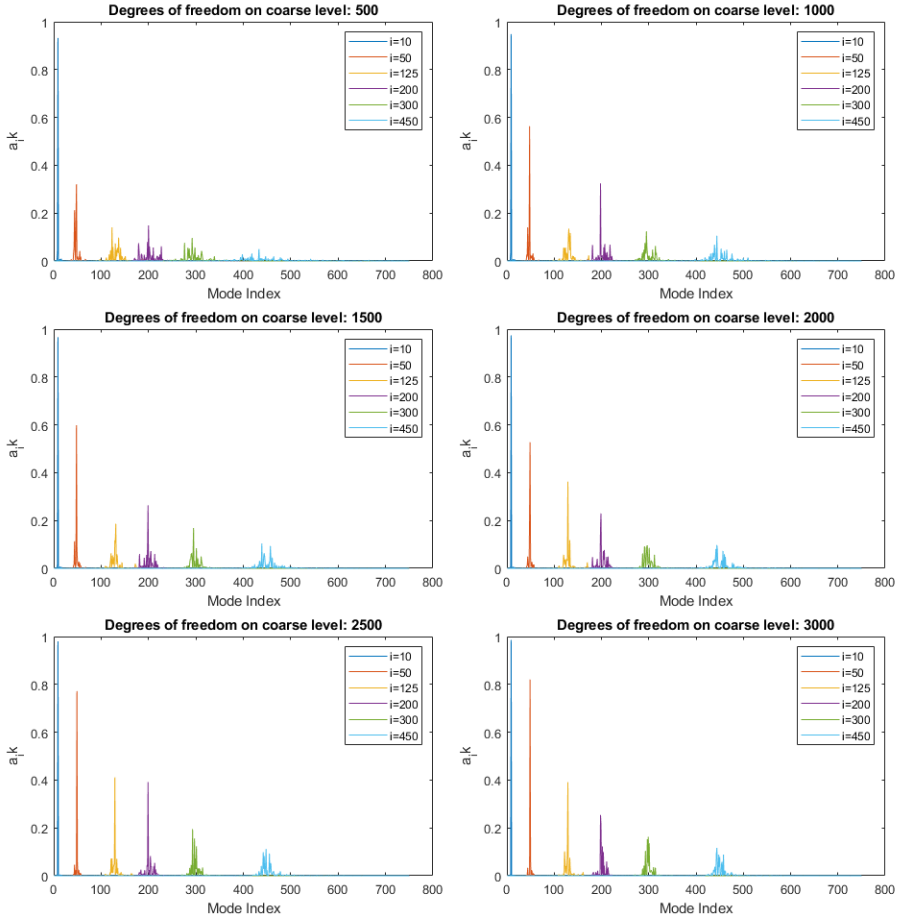


Figure 5.4: Magnitude of Fourier coefficients of the prolongated coarse level solution projected onto the reference solution for the rocker arm model

The number of degrees of freedom on the coarsest level shows a reduction in the number of iterations on the finest level but an increase in the amount of solve time. This can be explained by the higher hierarchy construction and dense level solve cost associated with a larger coarse level. This cost depends on the size of the problem as well as the characteristics of the object. Taking a look at figure 5.3 shows the Fourier coefficients of the 500Th mode. the graph shows that all Fourier coefficients are present for coarse level sizes above 3000. However, when looking at figure 5.4 we can see continuous improvement for higher coarse level sizes. We decided to settle on a minimum coarse level size of 3000 for our testing. However, we suggest that one might be able to make the case to increase this size if they can still efficiently solve the dense level when increasing its size.

5.5.1 Number of Levels

	Levels	2		3		4	
Object	Eigvals	its	Time	Its	Time	Its	Time
Rockerarm (∼56k DoF)	50	$F\|3$	8.9	$F\|3\|3$	21.1	$F\|2\|2\|3$	23.2
	500	$F\|7$	46.5	$F\|4\|5$	69.0	$F\|5\|5\|5$	140.0
Armadillo (∼693k DoF)	50	$F\|3$	80.3	$F\|2\|2$	81.9	$F\|2\|2\|2$	117.5
	250	$F\|5$	392.2	$F\|4\|4$	367.5	$F\|3\|3\|5$	527.6

Table 5.7: A table displaying the performance for different numbers of levels.

In this section, we compare the number of required iterations and the timings of the HSIM algorithm for varying numbers of levels. Table 5.7 shows the performance of HSIM for varying amounts of levels. The runs show a clear distinction between the armadillo object and the rockerarm object. The armadillo object sees an increase in performance when using 3 levels for 250 eigenpairs both in timing as well as iterations on the finest level. This differs for the rockerarm object, where a higher number of levels does not necessarily improve the timings. It does show an improvement in the number of iterations required on the finest level.

The number of levels to use when solving for a number of eigenvalues depends on the trade-off between a higher hierarchy construction cost as well as longer solve time for intermediate levels and the reduction of iterations on the finest level. The results show increased performance for a level count of 3 when the degrees of freedom of a model and the number of eigenpairs to be computed are large enough to allow for a gain in solve time when the number of iterations is reduced. Increasing the number of eigenvalues, as well as the degrees of freedom, increases the computation time per iteration on the finest level. Once this computation time is large enough to justify an extra level in order to reduce the number of required iterations on the finest level, we see a gain in solve time. This is not true to for the rockerarm model, where the degrees of freedom are small enough to justify the extra iterations instead of the increased level count.

5.6 Shifting

In this section, we compare the performance of varying shifting values α. In section 4.4 we state that an optimal shifting value should guarantee an overall optimal best performance. Looking at the results in table 5.8 shows that optimal value α differs per object per number of eigenpairs calculated. A value of α between 0.15 and 0.2 shows the best performance for the rockerarm object. Where the value $\alpha = 0.15$ guarantees the best performance in a time-based manner while keeping the required number of iterations on the finest level as low as possible.

The shifting value used by HSIM determines how far the stiffness matrix will be shifting into the unconverged eigenvalue range. HSIM can use the results of the previous level to determine the required shifting value. In our results, we can see different performances

	Shift value	$\alpha = 0$		$\alpha = 0.1$		$\alpha = 0.15$		$\alpha = 0.2$		$\alpha = 1/3$		$\alpha = 0.4$	
Object	Eigvals	Its	Time	Its	Time	Its	Time	Its	Time	Its	Time	Its	Time
Rockerarm (∼56k DoF)	50	$F\|2$	7.3	$F\|2$	7.5	$F\|2$	7.6	$F\|2$	7.0	$F\|3$	8.4	$F\|4$	10.1
	500	$F\|3$	29.4	$F\|3$	28.1	$F\|3$	27.8	$F\|4$	33.1	$F\|7$	46.5	$F\|10$	57.8
Armadillo (∼693k DoF)	50	$F\|3$	79.8	$F\|3$	80.3	$F\|3$	83.9	$F\|3$	87.0	$F\|3$	83.0	$F\|3$	80.6
	250	$F\|7\|5$	456.1	$F\|7\|5$	451.9	$F\|6\|5$	442.5	$F\|6\|5$	445.0	$F\|6\|4$	410.2	$F\|5\|5$	446.8

Table 5.8: A table showing the performance differences for different shifting amounts α.

for different meshes. The rocker arm mesh has a low number of iterations for all shifting values of $\alpha = 0.15$ or lower. This is different for the armadillo mesh, where the optimal performance can be seen at $\alpha = 1/3$.

5.7 Subspace Size

In this section, we compare the performance of HSIM with varying subspace sizes. In this experiment, we look at the iterations as well as the timings. In section 4.5 we discuss the goal of reducing the number o,f required iterations on the finest level to 1. This experiment will, therefore, highlight an optimal value for timing performance as well as for iteration count. The data in table 5.9 shows a strong decrease in the number of required iterations for

	Subspace size	1.5		2		2.5		3		3.5		4	
Object	Eigvals	Its	Time	Its	Time	Its	Time	Its	Time	Its	Time	Its	Time
Rockerarm (∼56k DoF)	50	$F\|2$	7.1	$F\|2$	7.4	$F\|2$	7.9	$F\|2$	8.5	$F\|2$	9.1	$F\|2$	9.4
	500	$F\|3$	27.6	$F\|3$	36.0	$F\|3$	45.9	$F\|5$	55.4	$F\|3$	67.2	$F\|4$	78.2
Armadillo (∼693k DoF)	50	$F\|3$	78.9	$F\|3$	90.4	$F\|2$	82.2	$F\|2$	95.0	$F\|2$	103.8	$F\|2$	117.4
	250	$F\|6\|5$	425.5	$F\|3\|4$	458.6	$F\|2\|3$	459.2	$F\|2\|4$	674.0	$F\|2\|4$	811.8	$F\|2\|4$	958.0

Table 5.9: A table showing the performance differences for different subspace size numbers.

the armadillo object for a subspace size of 2.5. This reduction is not seen for the rockerarm object, where the values are optimal for all subspace sizes 1.5-2.5.

Increasing the subspace size increases the cost of each iteration of SIM, however, it allows for a reduction in iterations on the finest level. A distinction can be made here when determining an optimal value. If we target minimizing the number of iterations on the finest level, we should use a subspace size of $q = 2.5$. However, if we aim to minimize compute time, we should use a subspace size of $q = 1.5$.

5.8 Tolerance

In this section, we compare the iteration and timings for both HSIM and SIM for different tolerances. Table 5.10 shows the results of these tests. Decreasing the tolerance shows a general increase in computation time as well as the number of iterations for both HSIM and SIM. Our previous runs with Matlab's Lanczos showed no need to compare timings between differing tolerances. The algorithm produces tolerances around $1e-5$ in most cases where it converged to $1e-2$. Lanczos is, in general, better tailored when computing for small tolerances. These results follow the statements made in the HSIM paper by Nasikun

Object	Tolerance	Eigvals	HSIM		SIM	
Armadillo (∼693k DoF)	1e-2	50	$F\|3$	74.6	8	314.5
		250	$F\|6\|5$	394.3	12	971.1
Rockerarm (∼56k DoF)		50	$F\|2$	6.9	9	15.2
		250	$F\|3\|3$	28.4	11	37.2
Armadillo (∼693k DoF)	1e-4	50	$F\|6$	111.1	14	386.8
		250	$F\|11\|11$	728.6	24	1742.0
Rockerarm (∼56k DoF)		50	$F\|6$	12.7	18	27.2
		250	$F\|6\|6$	49.5	23	73.4

Table 5.10: A table showing the difference in performance between HSIM and SIM for different tolerance values.

and Hildebrandt [2022]. A larger table with more models is available in appendix A as Table A.1.

5.9 Materials

In this section, we measure the potential impact that material parameters have on the performance of HSIM. In this section, we test with multiple material parameters. The material parameters used in this section are:

$$\text{Steel} \qquad k = 200GPa, \qquad v = 0.27, \qquad \text{density} = 7750kg/m^3 \tag{5.1}$$

$$\text{Rubber} \qquad k = 0.01GPa, \qquad v = 0.001, \qquad \text{density} = 1230kg/m^3 \tag{5.2}$$

For our first experiment, we compared the performance of HSIM and SIM for multiple models for both Steel and Rubber. The results of this experiment can be seen in table 5.11.

		Steel				Rubber			
Object	Eigvals	HSIM		SIM		HSIM		SIM	
		Its	Solve time	Its	Solve time	Its	Solve time	Its	Solve time
Armadillo (∼693k DoF)	50	$F\|3$	74.6	8	314.5	$F\|3$	70.2	8	314.5
Fandisk (∼28k DoF)		$F\|2$	6.0	8	14.0	$F\|2$	5.9	8	14.1
Rockerarm (∼56k DoF)		$F\|2$	6.9	9	15.2	$F\|2$	6.8	9	15.5
Sphere (∼182k DoF)		$F\|6$	39.5	10	1202.8	$F\|6$	38.5	10	1190.6
Spot (∼28k DoF)		$F\|3$	6.6	9	10.8	$F\|3$	6.7	9	10.9
Stepper body (∼33k DoF)		$F\|2$	7.0	9	10.5	$F\|2$	5.8	9	10.6

Table 5.11: A table showing the performance of HSIM and SIM for both steel and rubber material parameters.

The table shows no difference in the number of required iterations. It did, however, impact the timings of both algorithms. This can be explained by the change in matrix condition that occurs for the stiffness- and mass matrices when constructing them with other material parameters.

We also ran the experiment for a range of values for both Young's modulus as well as Poisson's ratio. The density was set at density $= 1000kg/m^3$ during the experiment as well. The results can be seen in Table 5.12.

	Young's modulus					
Poisson's ratio	0.01 GPa	0.1 Gpa	1 Gpa	10 Gpa	100 Gpa	1000 Gpa
0.0001	71.7	70.4	71.2	70.4	70.4	71.5
0.1	71.7	70.1	70.2	70.4	69.7	71.8
0.2	69.8	70.0	69.1	69.7	69.6	69.7
0.3	69.0	69.5	69.3	69.2	69.4	69.3
0.4	69.2	69.3	69.2	69.6	69.4	69.8
0.49	70.1	69.3	69.5	69.6	69.4	69.5

Table 5.12: A table showing the timings of computing 50 eigenpairs for the armadillo mesh with varying material parameters.

The table shows a general trend towards lower compute timings for a Poisson's ratio larger than 0.1. However, the general difference in timing is small enough to consider it to have little impact on testing results as long as the same parameters are used between tests.

5.10 Turning Vectors

In this section, we will discuss the data of the experiments discussed in section 4.8. In this section, we discussed the position of the enhanced SIM algorithm. We discuss the option to use the enhanced SIM algorithm as an enhancement method for HSIM, and we discuss whether or not it should be considered as a competing algorithm, suggesting we should benchmark against it.

The data in this section will first show the performance of HSIM when using the enhanced SIM algorithm in one of the three configurations discussed in 4.8. It will then compare the performance of HSIM to the performance of enhanced SIM itself to establish the performance between the two.

5.10.1 With Hierarchy

		HSIM turning on finest			HSIM turning on all levels			HSIM turning on intermediate		
Object	Eigvals	Its	# Turns	Solve time	Its	# Turns	Solve time	Its	# Turns	Solve time
Rockerarm (∼56k DoF)	50	$F\|1+1$	0	8.0	$F\|1+1$	0	7.9	$F\|1+1\|2$	0	13.1
	250	$F\|1+3$	0	26.3	$F\|1+3$	0	28.1	$F\|1+1\|2$	0	33.5
	500	$F\|1+4$	0	65.5	$F\|1+4$	0	64.6	$F\|1+2\|3$	0	83.5
Armadillo (∼693k DoF)	50	$F\|1+2$	5	88.8	$F\|1+2$	3	91.7	$F\|1+1\|2$	0	98.9
	250	$F\|2\|1+3$	0	460.2	$F\|1+2\|1+3$	0|0	456.7	$F\|1+2\|3$	0	463.0
	500	$F\|3\|1+3$	0	1257.4	$F\|1+2\|1+4$	0|1	1244.8	$F\|1+2\|4$	0	1232.6

Table 5.13: A table comparing the differences in performance when applying SIM with turning vectors to certain levels in HSIM against HSIM and the enhanced SIM algorithm. The "1+" symbolises the required normal SIM step before using enhanced SIM.

In the table 5.13, we compare the performance of the three HSIM configurations discussed in section 4.8. Every configuration lists the required iterations for convergence, the number of turns that enhanced SIM executed for the respective levels and the total solve

time. We decided to put the minimum amount of levels for the third configuration, ”HSIM turning on intermediate”, on 3 to ensure the presence of intermediate levels.

The notation of the number of iterations in table 5.13 is a small deviation from the notation described in section 5.2.4. We decided to add a ”1+” to the number of iterations for the levels where enhanced SIM was used. The method requires an initial SIM iteration before the turning iterations can be started. This brings us to the number of turns. Making use of enhanced SIM together with the hierarchy results in almost no turns. The rockerarm model did not require any turns at all. While the armadillo mesh did require turning for some runs, it did not result in significant performance improvements.

Looking at the performance of using enhanced SIM in conjunction with HSIM, we can see no performance increase compared to normal HSIM. Enhanced SIM has little use within the hierarchy when we look at the number of turns taken by the method. The algorithm reverts to a normal SIM algorithm with some overhead when no turns are made. Explaining the small dip in performance.

5.10.2 Without Hierarchy

		Enhanced SIM			HSIM	
Object	Eigvals	Its	# Turns	Solve time	Its	Solve time
Rockerarm (∼56k DoF)	50	1+4	54	10.1	$F\|2$	7.9
	250	1+6	393	43.9	$F\|3$	24.9
	500	1+7	824	120.1	$F\|3$	45.9
Armadillo (∼ 693k DoF)	50	1+4	40	95.6	$F\|2$	82.22
	250	1+6	365	687.8	$F\|2\|3$	459.2
	500	1+7	759	2513.8	$F\|3\|3$	1064.4

Table 5.14: A table comparing the differences in performance between HSIM and the enhanced SIM algorithm. The ”1+” symbolises the required normal SIM step before using enhanced SIM.

The data displayed in section 5.10.1 shows the performance for HSIM when applying the enhanced SIM algorithm in varying configurations. In this section, we will compare the performance of HSIM against enhanced SIM itself. Table 5.14 compares the performance of enhanced SIM versus HSIM. The amount of turning vectors used by enhanced SIM differ significantly compared to the numbers shown in table 5.13. Putting both algorithms side to side we can see that HSIM beats enhanced SIM on the amount of required iterations on the finest level as well as in total. This pattern is also visible in the timings as well. The difference between HSIM and Enhanced SIM becomes clear when we look at table 5.13. Enhanced SIM is using significantly more turns when used as a stand-alone method. These turns result in a relatively fast solve time compared to normal SIM. This speed-up seems to lose its effectiveness when computing a large amount of eigenpairs. Our implementation of the algorithm tries to parallelize as much of the algorithm as possible, which differs from the initial implementation shown by Kim and Bathe [2017]. The vector turning is partially

single threaded, which might explain the poorer scaling when the number of eigenpairs increases. This would require more research to say this with certainty.

5.11 Scaling

This section focuses on the relative scaling of HSIM, SIM and Matlab. We will evaluate the scaling of the algorithms for two criteria:

- Number of eigenvalues
- Number of cores (parallel scaling)

It is important to note the differences between the SIM and HSIM implementation in this section. The SIM algorithm makes use of Eigen's SimplicialLDLT solver as HSIM makes use of Intel's Pardiso solver. Our testing shows that Intel's Pardiso solver is faster during factorization but at the cost of longer solve timings compared to Eigen's SimplicialLDLT solver. There was no clear source to confirm that this was expected behaviour apart from some Github repositories that included benchmarks[1]. Using SIM together with Intel's Pardiso solver resulted in instabilities during benchmarking, which would prevent some benchmarks from yielding results. Which is why we use Eigen's SimplicialLDLT for SIM.

5.11.1 Number of Eigenvalues

In this section, we evaluate the performance scaling of HSIM, SIM and Matlab when increasing the amount of eigenpairs to compute. For our testing, we decided to focus on scaling the number of Eigenvalues for the rockerarm problem.

Object	Eigvals	HSIM		SIM		Matlab
		Its	Solve time	Its	Solve time	Solve time
Rockerarm (∼56k DoF)	50	$F\|2$	7.6	5	12.0	2.3
	250	$F\|3$	16.8	7	39.0	12.1
	500	$F\|3$	32.1	8	99.0	41.2
	1000	$F\|4$	63.4	8	193.6	126.9
	2000	$F\|4$	178.4	7	441.5	430.6
	3000	$F\|5$	369.2	7	811.7	910.0
	4000	$F\|5$	514.2	7	1367.4	1522.0
Armadillo (∼693k DoF)	50	$F\|2$	78.6	5	352.4	43.2
	250	$F\|2\|3$	367.5	7	1208.2	212.0
	500	$F\|3\|3$	917.5	8	2572.3	680.9

Table 5.15: A table showing the iterations and timings of HSIM, SIM and Matlab's Lanczos solver.

[1] Both repositories look at the performance and tolerance of different solvers:`https://github.com/alecjacobson/sparse-solver-benchmark`, `https://github.com/BeanLiu1994/solver_speed_test`

Table 5.15 shows the number of iterations and the timings for HSIM and SIM, as well as the timings for Matlab's Lanczos solver. The table shows both HSIM and SIM closing the gap with the Lanczos solver. HSIM is able to outperform the Lanczos solver when the eigenpair count rises above 500. SIM eventually beats the Lanczos solver when the number of eigenpairs reaches 3000.

5.11.2 Parallel Scaling

In this section, we evaluate the relative parallel scaling performance of HSIM and SIM. We aim to determine the performance characteristics when applying these methods on high-core count computers. All tests were run on the test system described in section 5.1

Threads	HSIM			SIM		
	50	250	500	50	250	500
	Solve Time	Solve Time	Solve Time	Solve Time	Solve Time	Solve Time
1	11.5	47.8	106.7	34.6	157.5	350.0
2	8.6	29.1	58.1	22.8	86.4	184.9
4	7.5	19.9	41.8	18.6	57.0	114.9
6	7.1	17.0	31.0	16.8	44.6	84.9
8	7.0	16.0	26.4	15.9	38.9	76.5
10	7.0	18.4	26.9	16.2	42.0	76.7
12	7.0	16.4	27.0	16.1	40.2	72.3
14	7.0	16.7	26.8	15.8	38.2	70.7
16	7.6	16.8	32.1	16.1	40.4	75.2

Table 5.16: A table containing the runtimes and iteration count of HSIM and SIM for the rocker arm problem with varying core counts. The computer used for the benchmarks has 8 physical cores and simultaneous multi-threading.

Table 5.16 shows the performance of HSIM and SIM for different thread counts. Each method is tested against the rocker arm problem 3 times, with each run increasing the number of computed eigenpairs. The table lists 16 threads. 8 of which are hardware threads, and the other 8 are produced by the Simultaneous Multi-Threading[2] implementation of our processor. The operating system's thread scheduling prioritizes hardware threads, which means the runs with 8 or fewer threads have a physical CPU core available for each thread. This changes when increasing the thread count to 9 or above. This information is important since it highlights the difference between HSIM and SIM. Where HSIM reaches its fastest timing on the 8Th thread. SIM does so on the 16Th thread. Although with a small gain. HSIM does seem to show the same behaviour when increasing the number of eigenpairs to calculate, leading us to believe that the iterations themselves benefit from virtual threads. Simultaneous Multi-Threading does not guarantee increased performance when both threads

[2]A technique for assigning multiple independent threads of execution to one CPU core: `https://en.wikipedia.org/wiki/Simultaneous_multithreading`

are working on the same resources, as shown by Qun et al. [2016]. It would be better to evaluate performance scaling on strictly physical cores to get a better comparison.

Cores	HSIM			SIM		
	50	250	500	50	250	500
	Solve Time	Solve Time	Solve Time	Solve Time	Solve Time	Solve Time
1	11.5	47.8	106.7	34.6	157.5	350.0
2	8.6	29.1	58.1	22.8	86.4	184.9
3	8.0	23.2	46.5	19.7	66.6	136.1
4	7.5	19.9	41.8	18.6	57.0	114.9
5	7.4	18.6	32.6	17.6	50.4	97.7
6	7.1	17.0	31.0	16.8	44.6	84.9
7	7.1	16.6	27.8	16.5	41.1	77.4
8	7.0	16.0	26.4	15.9	38.9	76.5

Table 5.17: A table containing the runtimes and iteration count of HSIM and SIM for the rocker arm problem for varying amounts of physical cores.

Table 5.17 shows the performance of HSIM and SIM for all core counts between 1 and 8. This data allows us to graph the relative performance gain for each number of cores compared to executing the methods sequentially. Graphing the performance in figure 5.5 shows a similar curve for both methods.

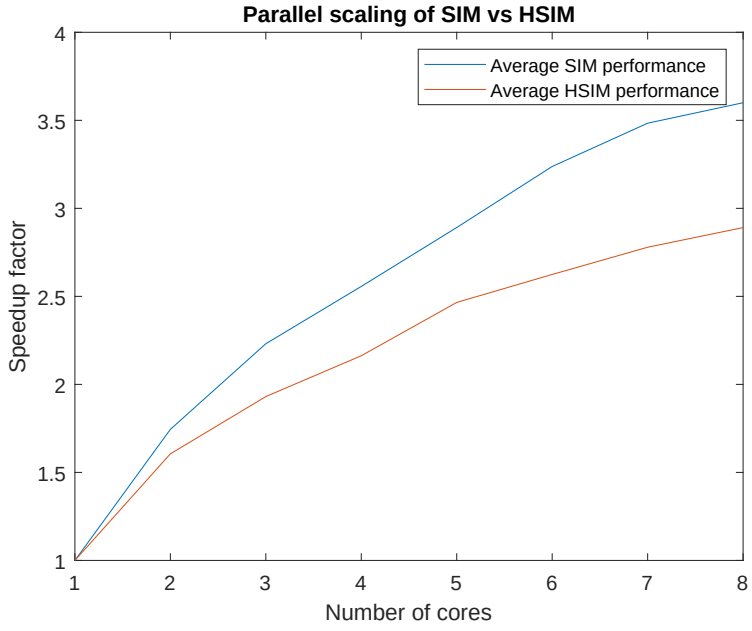


Figure 5.5: A graph showing the factor of speedup compared to single core performance per number of cores

HSIM does show a faster decrease in performance gain when increasing core count.

This can be attributed to the extra computational steps that are taken by HSIM during hierarchy construction and coarse-level solving. The addition of these steps causes poorer scaling per number of cores for HSIM compared to SIM. Nevertheless, the scaling of both SIM and HSIM show the same reduction in effectiveness when comparing both trends in figure 5.5. In practice, we can say that HSIM is fast enough to mitigate this relative scaling issue. Looking at the parallel performance of HSIM compared to SIM, we can see that both algorithms scale well when increasing core count. The HSIM algorithm reaches its optimal performance at 8 threads while the SIM algorithm keeps improving its solve time up to 16 threads. This indicates that SIM enjoys more benefit from multi-threading compared to HSIM. Multi-threading allows the processor to execute instructions from another thread while it waits for instructions from the current thread. This can speed up calculations. However, depending on the operations being executed, it might also negatively impact calculation speed. Qun et al. [2016] show that hyper-threading has little effect on parallel CPU-bound workflows.

This prompted us to look at the scaling of both algorithms on physical cores only. Looking at figure 5.5 shows the speedup of HSIM per number of cores following the trend of SIM up until a core count of 5. Where the incremental speedup of HSIM seems to decline compared to SIM. The hierarchy construction of HSIM is done in parallel whenever possible however, it does contain serial code. This contributes to a larger percentage of serial code for HSIM compared to SIM. That does not indicate that SIM will be faster than HSIM when increasing the number of cores. HSIM reduces the overall amount of solve time by a significantly large margin that SIM would require a factor > 2 of better scaling performance before it catches up to HSIM.

5.12 Multiple models

This section includes a table with testing results for a multitude of models to demonstrate the performance of HSIM compared to SIM and Matlab's Lanczos solver on a larger set of models. The table shows Matlab's Lanczos solver being faster than HSIM in most cases.

		HSIM		SIM		Matlab's Lanczos
Object	Eigvals	Its	Solve time	Its	Solve time	Solve time
Armadillo (∼693k DoF)	50	$F\|3$	74.6	8	314.5	42.7
	250	$F\|6\|5$	394.3	12	971.1	201.5
Blade smooth (∼1.3M DoF)	50	$F\|4$	197.0	DNF	DNF	120.6
	250	$F\|6\|7$	961.0	DNF	DNF	519.6
Fandisk (∼28k DoF)	50	$F\|2$	6.0	8	14.0	2.0
	250	$F\|2\|4$	20.5	12	36.0	12.1
Rockerarm (∼56k DoF)	50	$F\|2$	6.9	9	15.2	2.3
	250	$F\|3\|3$	28.4	11	37.2	11.8
Sphere (∼182k DoF)	50	$F\|6$	39.5	10	1202.8	53.4
	250	$F\|2\|12$	167.9	12	1894.5	147.7
Spot (∼28k DoF)	50	$F\|3$	6.6	9	10.8	1.2
	250	$F\|3\|3$	17.8	13	25.2	8.1
Stepper mount (∼33k DoF)	50	$F\|2$	7.0	9	10.5	1.4
	250	$F\|2\|3$	16.5	10	19.7	6.2

Table 5.18: A table showing the performance of HSIM, SIM and Matlab's Lanczos solver for multiple models. The tolerance is set to 1e-2 for all algorithms. DNF stands for "Did Not Finish" where the algorithm did not converge.

HSIM does win in one case when computing 50 eigenpairs for the Sphere model. This can be explained by the topology of the Sphere mesh, which is fully symmetric.

Below, we include an expansion on the results shown in Table 5.18 where we compare HSIM, SIM and Matlab's Lanczos solver against each other when computing 750 eigenpairs. These results are run on a slightly different system where the CPU in the system is replaced by an AMD Ryzen 9 5950X 16-core processor. This is why we include these results in a separate table. The results can be seen in Table 5.19.

These results show the strength of HSIM when focusing on a larger number of eigenpairs.

Object	Eigvals	HSIM		SIM		Matlab's Lanczos
		Its	Solve time	Its	Solve time	Solve time
Armadillo (∼693k DoF)	750	$F\|4\|5$	713.5	13	1403.2	739.6
Fandisk(∼28k DoF)		$F\|3\|4$	35.6	13	53.1	50.3
Rockerarm (∼56k DoF)		$F\|3\|3$	41.7	13	61.1	49.1
Sphere (∼182k DoF)		$F\|3\|12$	324.8	12	1880.8	625.5
Stepper body (∼33k DoF)		$F\|5$	33.9	13	36.4	46.0

Table 5.19: A table showing the results of HSIM, SIM and Matlab's Lanczos solver for a subset of the testing models computing 750 eigenpairs. These tests were run on a slightly different system where the processor was replaced with an AMD Ryzen 9 5950X 16-core processor. All other components remain the same. The tolerance was set to 1e-2.

Chapter 6

Conclusion and future work

This chapter gives an overview of the project's contributions. After this overview, we will reflect on the results and conclude. Finally, some ideas for future work will be discussed.

6.1 Discussion

The first goal of this work was to determine if it was possible to apply HSIM to vibration mode problems. This required an adaptation to HSIM to work with volumetric meshes for the hierarchy. The second goal was to find out if it was possible to increase the performance by finding well-performing parameters and optimizing hierarchy construction for the problem specifically. As can be seen in this work, HSIM was adapted to the problem of vibration modes, and a search for optimizations and parameters has been done.

Our results show a strong improvement in computation time when comparing HSIM and SIM. HSIM manages to reduce the number of iterations required on the finest level for every problem included in our experiment setup while reducing the compute time by a minimal factor of 1.5 while showing a median improvement of factor 2 compared to SIM. The best improvement factor measured during our testing was a factor of 30 for computational speed. Our sphere model caused some instability for SIM, where the factorization took significantly longer. This issue did not impact HSIM, which can be explained by the shifting that HSIM uses.

An important point to note when evaluating the results of our experiments is the approach to parameter tuning. We decided to test the parameters in order of dependency on each other, hoping that intermediate results would allow us to tune for the next test better. However, this approach is limiting in terms of the exploration of total performance. It might, for example, be possible that the results in Section would differ greatly when a different number of levels would be used. This goes for other results as well, where testing all possible combinations is considered too heavy on computing resources and time.

Another point of discussion is the selected models used during testing. testing a wide range

of models is limiting in the number of parameters we could test for, which resulted in a lot of two-model tables in the results. Every algorithm responds differently to different models as well as different volumetric meshes made from the same model. This creates an entirely different problem where one would have to grade the volumetric mesh that is used to test. We think that separate research investigating the performance of HSIM when compared to multiple meshes of differing conditions would be beneficial. This is why we recommend that in the future work section6.3.1 Comparing the performance of HSIM against enhanced SIM yields interesting results. Using enhanced SIM together with the hierarchy results in little to no turns for the entire computation. Running enhanced SIM on the same problem without the hierarchy shows the opposite, where a lot of turns are made during the computation. The difference in number suggests that HSIM and enhanced SIM both optimise the same thing. They both optimise the iteration vectors, with the difference being that HSIM optimizes the iteration vectors as a pre-computation in the form of a hierarchy. Enhanced SIM optimizes the iteration vectors every iteration when the algorithm determines that turning is needed.

The relative performance improvement of HSIM compared to SIM in our experiments is smaller than the improvement shown in the original HSIM paper by Nasikun and Hildebrandt [2022]. Both in timings as well as a number of iterations. A possible explanation can be the density of the problem, which impacts the speed of factorization for the layers as well as the solving step for each iteration. A faster factorization library for our specific problem might increase performance for both HSIM and SIM, bringing it closer to Matlab's Lanczos solver. This should be investigated further, which we recommend in the future work section 6.3.2.

6.2 Conclusion

Our study presents an adaptation of the Hierarchical Subspace Iteration Method tailored for volumetric mesh-based generalized eigenproblems, where we focus on computing vibration modes. Our evaluation shows that HSIM performs significantly better than SIM in all cases and even Matlab's Lanczos solver in some cases. The effectiveness of the hierarchy has been analyzed and shown to be capable of strongly reducing the number of required iterations to converge HSIM.

In conclusion, our adaptation of the Hierarchical Subspace Iteration Method by Nasikun and Hildebrandt [2022] shows a strong improvement over existing SIM-based methods for computing vibration modes of elastic meshes where computation time is improved by a factor of 1.5-2.5 for all problems.

Our work shows that HSIM is a better choice overall than SIM when computing vibration modes. It results in a faster computation with reduced iteration count while maintaining its strong parallel scaling and stability. The hierarchy is capable of representing the problem

on a coarser level, allowing for proper prolongation independent of material parameters and matrix conditions. However, HSIM is still not the best choice when computing for a small tolerance or small number of eigenpairs. Therefore, we propose a few directions for future work on HSIM. Further research in these directions might allow for improvements to HSIM that would allow it to outperform all existing methods.

6.3 Future work

In this section, we discuss directions for future work that could, in our opinion, improve HSIM.

6.3.1 Matrix conditioning

During our exploratory phase, we encountered varying performances of HSIM while running the same mesh. We would tetrahedralize the mesh for every run with tetgen, but we did not configure any strong quality requirements, which resulted in matrices of varying condition numbers. The condition number of a matrix is shown to influence the convergence of iterative methods as shown by Pyzara et al. [2011]. The matrices associated with our models had condition numbers between 10^5 to 10^7 where we could see the lower condition number in better iteration reduction for HSIM.

6.3.2 Factorization performance

HSIM makes use of another factorization algorithm than the SIM algorithm. This was already mentioned in section 5.11. SIM is not stable when using the Pardiso solver. Finding a proper source that is capable of answering why this is the case is next to impossible. However, the phenomenon is recorded in multiple Github repositories where multiple libraries are pitted against each other[1]. The author of the repository states that the Eigens SimplicialLDLT solver struggles with less sparse matrices. His example of a less sparse matrix contains around 40 non-zero rows.

The repository also shows the error norm for the different benchmarks he executed on his machine. It shows Intels Pardiso solver's result having a 3 times greater L∞ norm than Eigens SimplicialLDLT. This is all derived from this GitHub repository, which is why it requires further research.

Researching this problem could help create a better understanding of which factorization to use for HSIM. As mentioned before, SIM showed unstable behaviour when using the Pardiso solver. The first iteration would crash the dense solver, suggesting that our projected matrix was not positive semi-definite. The behaviour was sporadic, suggesting that it partially depended on the random initialisation of the iteration vectors. The vectors were checked on orthogonality. HSIM did not suffer from this instability and can make use of both solvers. Suggesting that the hierarchy prevents this behaviour by providing better-conditioned iteration vectors.

[1] A repository showing benchmarks of sparse solvers: `https://github.com/alecjacobson/sparse-solver-benchmark`

6.3.3 Material based basis construction

The current basis construction of HSIM for vibration modes does not use the material properties for the sampling of the vertices or determine the influence of vertices around the sample. It might be worthwhile to research the possible impact of optimizing the basis construction for vibration modes with the use of material properties. The basis construction can be tailored for two different problem areas where the algorithm can be adapted in multiple places.

The first problem area would focus on homogeneous materials where the parameters of the basis construction are computed based on the material properties. This would require extensive parameter searching for a multitude of models, but it might result in an easier-to-use algorithm with fewer parameters to set for the user.

The second problem area would focus on composite materials where the material parameters differ along the mesh. These problems might benefit from material-based basis construction more than the homogeneous approach. A model with strongly different material parameters on both ends could possibly benefit from different sampling rates along the mesh.

For the basis construction based on material properties, we should consider the difference between isotropic materials and anisotropic materials. An Isotropic material is a direction invariant to its response to applied force, while an anisotropic material is not. A common example of an isotropic material is rubber, and for an anisotropic material, it is wood.

Isotropic materials could make use of a basis construction where we treat the 3 degrees of freedom per vertex all equally. We can make use of the density, Poisson's ratio and Young's modulus for both the sampling as well as determining the possible influence on its neighbours.

Anisotropic materials could benefit from a different basis construction where we treat every degree of freedom as possible standalone samples. Materials might be less elastic in one axis compared to other axis'. The basis construction could account for that and sample differently for each axis. Determining the effects of this method of sampling would require extensive testing and exploration, in our opinion. Nevertheless, we think that anisotropic materials might benefit the most from a tailored basis construction. Since it might allow a reduction in the degrees of freedom on the coarse level deemed necessary for a good translation to the finest level.

We think the basis construction of HSIM for vibration modes could benefit from specialization based on material properties. We can only theorize about the best method adaptations, which is why we listed a few in this section. The actual benefits should be explored in a separate future research.

Bibliography

Klaus-Jürgen Bathe and Seshadri Ramaswamy. An accelerated subspace iteration method. *Computer Methods in Applied Mechanics and Engineering*, 23(3):313–331, 1980a.

Klaus-Jürgen Bathe. The subspace iteration method – revisited. *Computers & Structures*, 126:177–183, 2013. ISSN 0045-7949. doi: https://doi.org/10.1016/j.compstruc.2012.06.002. URL https://www.sciencedirect.com/science/article/pii/S0045794912001460. Uncertainty Quantification in structural analysis and design: To commemorate Professor Gerhart I. Schueller for his life-time contribution in the area of computational stochastic mechanics.

Klaus-Jürgen Bathe and Seshadri Ramaswamy. An accelerated subspace iteration method. *Computer Methods in Applied Mechanics and Engineering*, 23(3):313–331, 1980b. ISSN 0045-7825. doi: https://doi.org/10.1016/0045-7825(80)90012-2. URL https://www.sciencedirect.com/science/article/pii/0045782580900122.

Klaus-Jürgen Bathe and Edward L. Wilson. Solution methods for eigenvalue problems in structural mechanics. *International Journal for Numerical Methods in Engineering*, 6(2):213–226, 1973. doi: https://doi.org/10.1002/nme.1620060207. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620060207.

James H Bramble, Joseph E Pasciak, and Andrew V Knyazev. A subspace preconditioning algorithm for eigenvector/eigenvalue computation. *Advances in Computational Mathematics*, 6:159–189, 1996.

Tsu-Chien Cheu, C Philip Johnson, and Roy R Craig Jr. Computer algorithms for calculating efficient initial vectors for subspace iteration method. *International journal for numerical methods in engineering*, 24(10):1841–1848, 1987a.

Tsu-Chien Cheu, C. Philip Johnson, and Roy R. Craig Jr. Computer algorithms for calculating efficient initial vectors for subspace iteration method. *International Journal for Numerical Methods in Engineering*, 24(10):1841–1848, 1987b. doi: https://doi.org/10.1002/nme.1620241003. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620241003.

Nguyen Chi Tho, Do Van Thom, Pham Hong Cong, Ashraf M. Zenkour, Duc Hong Doan, and Phung Van Minh. Finite element modeling of the bending and vibration behavior of three-layer composite plates with a crack in the core layer. *Composite Structures*, 305:116529, 2023. ISSN 0263-8223. doi: https://doi.org/10.1016/j.compstruct.2022.116529. URL https://www.sciencedirect.com/science/article/pii/S0263822322012612.

Keenan Crane, Ulrich Pinkall, and Peter Schröder. Robust fairing via conformal curvature flow. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013.

Jane K Cullum and Ralph A Willoughby. *Lanczos algorithms for large symmetric eigenvalue computations: Vol. I: Theory*. SIAM, 2002.

G D Dean and L E Crocker. The use of finite element methods for design with adhesives. Measurement Good Practice Guide 10.47120/npl.mgpg48, Teddington, March 2023. URL http://eprintspublications.npl.co.uk/2770/.

Marco Fratarcangeli, Valentina Tibaldo, Fabio Pellacini, et al. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.*, 35(6):214–1, 2016.

Rachel Gaal and Alaina G. Levine. November 7, 1940: Collapse of the tacoma narrows bridge, Nov 2016. URL https://www.aps.org/publications/apsnews/201611/physicshistory.cfm.

Geuzaine, Christophe and Remacle, Jean-Francois. Gmsh. URL http://http://gmsh.info/.

Luc Giraud, Julien Langou, and Miroslav Rozloznik. The loss of orthogonality in the gram-schmidt orthogonalization process. *Computers & Mathematics with Applications*, 50(7):1069–1075, 2005.

Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

Stefan Gumhold, Stefan Guthe, and Wolfgang Straßer. Tetrahedral mesh compression with the cut-border machine. pages 51–58, 01 1999. doi: 10.1109/VISUAL.1999.809868.

N. Guo, P. Cawley, and D. Hitchings. The finite element analysis of the vibration characteristics of piezoelectric discs. *Journal of Sound and Vibration*, 159(1):115–138, 1992. ISSN 0022-460X. doi: https://doi.org/10.1016/0022-460X(92)90454-6. URL https://www.sciencedirect.com/science/article/pii/0022460X92904546.

Alec Jacobson and Daniele Panozzo. Libigl: Prototyping geometry processing research in c++. In *SIGGRAPH Asia 2017 Courses*, SA '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450354035. doi: 10.1145/3134472.3134497. URL https://doi.org/10.1145/3134472.3134497.

Ki-Tae Kim and Klaus-Jürgen Bathe. The bathe subspace iteration method enriched by turning vectors. *Computers & Structures*, 186:11–21, 2017. ISSN 0045-7949. doi: https://doi.org/10.1016/j.compstruc.2017.02.006. URL https://www.sciencedirect.com/science/article/pii/S0045794916314572.

R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. Available from netlib@ornl.gov, 1997.

RENFU LU and JUDITH A. ABBOTT. Finite element analysis of modes of vibration in apples. *Journal of Texture Studies*, 27(3):265–286, 1996. doi: https://doi.org/10.1111/j.1745-4603.1996.tb00075.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1745-4603.1996.tb00075.x.

Ahmad Nasikun and Klaus Hildebrandt. The hierarchical subspace iteration method for laplace–beltrami eigenproblems. *ACM Trans. Graph.*, 41(2), jan 2022. ISSN 0730-0301. doi: 10.1145/3495208. URL https://doi.org/10.1145/3495208.

Ahmad Nasikun, Christopher Brandt, and Klaus Hildebrandt. Supplementary material for fast approximation of laplace–beltrami eigenproblems.

Ahmad Nasikun, Christopher Brandt, and Klaus Hildebrandt. Fast approximation of laplace-beltrami eigenproblems. In *Computer Graphics Forum*, volume 37, pages 121–134. Wiley Online Library, 2018.

Anna Pyzara, Beata Bylina, and Jarosław Bylina. The influence of a matrix condition number on iterative methods' convergence. In *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 459–464, 2011.

Ng Hui Qun, Z.I.A Khalib, M. N. Warip, M. Elshaikh Elobaid, R. Mostafijur, N.A.H. Zahri, and Puteh Saad. Hyper-threading technology: Not a good choice for speeding up cpu-bound code. In *2016 3rd International Conference on Electronic Design (ICED)*, pages 578–581, 2016. doi: 10.1109/ICED.2016.7804711.

Martin Reuter, Franz-Erich Wolter, and Niklas Peinecke. Laplace–beltrami spectra as 'shape-dna' of surfaces and solids. *Computer-Aided Design*, 38(4):342–366, 2006. ISSN 0010-4485. doi: https://doi.org/10.1016/j.cad.2005.10.011. URL https://www.sciencedirect.com/science/article/pii/S0010448505001867. Symposium on Solid and Physical Modeling 2005.

Olaf Schenk and Klaus Gärtner. *PARDISO*, pages 1458–1464. Springer US, Boston, MA, 2011. ISBN 978-0-387-09766-4. doi: 10.1007/978-0-387-09766-4_90. URL https://doi.org/10.1007/978-0-387-09766-4_90.

Hang Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2), feb 2015. ISSN 0098-3500. doi: 10.1145/2629697. URL https://doi.org/10.1145/2629697.

Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids. 2012. doi: 10.1145/2343483.2343501.

Xuelin Wang and Ji Zhou. An accelerated subspace iteration method for generalized eigenproblems. *Computers & structures*, 71(3):293–301, 1999.

Bryce Daniel Wilkins. *The E^2 Bathe subspace iteration method*. PhD thesis, Massachusetts Institute of Technology, 2019.

Xice Xu, Yang Lu, Mengxue Shao, and Chunbo Lan. Fast prediction method for multirotor global tonal noise based on acoustic modal analysis. *Mechanical Systems and Signal Processing*, 183:109620, 2023. ISSN 0888-3270. doi: https://doi.org/10.1016/j.ymssp.2022.109620. URL https://www.sciencedirect.com/science/article/pii/S0888327022007099.

Xin Yan, Jianmin Zhang, Songping Wu, Ming-Feng Xue, Chi Kin Benjamin Leung, Eric A. MacIntosh, and Daryl G. Beetner. A methodology for predicting acoustic noise from singing capacitors in mobile devices. *IEEE Transactions on Electromagnetic Compatibility*, 65(4):1266–1270, 2023. doi: 10.1109/TEMC.2023.3280922.

Ryan M. Zbikowski and Calvin W. Johnson. Bootstrapped block lanczos for large-dimension eigenvalue problems. *Computer Physics Communications*, 291:108835, 2023. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2023.108835. URL https://www.sciencedirect.com/science/article/pii/S0010465523001807.

Qian-Cheng Zhao, Pu Chen, Wen-Bo Peng, Yu-Cai Gong, and Ming-Wu Yuan. Accelerated subspace iteration with aggressive shift. *Computers & Structures*, 85(19):1562–1578, 2007. ISSN 0045-7949. doi: https://doi.org/10.1016/j.compstruc.2006.11.033. URL https://www.sciencedirect.com/science/article/pii/S0045794907000405.

D.Y. Zheng and N.J. Kessissoglou. Free vibration analysis of a cracked beam by finite element method. *Journal of Sound and Vibration*, 273(3):457–475, 2004. ISSN 0022-460X. doi: https://doi.org/10.1016/S0022-460X(03)00504-2. URL https://www.sciencedirect.com/science/article/pii/S0022460X03005042.

Ömer Civalek and Hayri Metin Numanoğlu. Nonlocal finite element analysis for axial vibration of embedded love–bishop nanorods. *International Journal of Mechanical Sciences*, 188:105939, 2020. ISSN 0020-7403. doi: https://doi.org/10.1016/j.ijmecsci.2020.105939. URL https://www.sciencedirect.com/science/article/pii/S0020740320308067.

Appendix A

Tables

Object	Tolerance	Eigvals	HSIM		SIM	
Armadillo (∼693k DoF)	1e-2	50	$F\|3$	74.6	8	314.5
		250	$F\|6\|5$	394.3	12	971.1
Fandisk(∼28k DoF)		50	$F\|2$	6.0	8	14.0
		250	$F\|2\|4$	20.5	12	36.0
Rockerarm (∼56k DoF)		50	$F\|2$	6.9	9	15.2
		250	$F\|3\|3$	28.4	11	37.2
Sphere (∼182k DoF)		50	$F\|6$	39.5	10	1202.8
		250	$F\|2\|12$	167.9	12	1894.5
Spot (∼28k DoF)		50	$F\|3$	6.6	9	10.8
		250	$F\|3\|3$	17.8	13	25.2
Stepper body (∼33k DoF)		50	$F\|2$	7.0	9	10.5
		250	$F\|2\|3$	16.5	10	19.7
Armadillo (∼693k DoF)	1e-4	50	$F\|6$	111.1	14	386.8
		250	$F\|11\|11$	728.6	24	1742.0
Fandisk (∼28k DoF)		50	$F\|6$	11.6	17	25.3
		250	$F\|7\|8$	47.5	26	73.8
Rockerarm (∼56k DoF)		50	$F\|6$	12.7	18	27.2
		250	$F\|6\|6$	49.5	23	73.4
Sphere (∼182k DoF)		50	$F\|7$	50.0	22	1351.9
		250	$F\|7\|15$	251.8	26	2753.9
Spot (∼28k DoF)		50	$F\|6$	10.1	18	20.6
		250	$F\|9\|9$	42.7	28	53.5
Stepper body (∼33k DoF)		50	$F\|5$	9.5	16	18.5
		250	$F\|6\|6$	36.2	20	39.0

Table A.1: A table showing all the results of running SIM and HSIM with differing tolerances for multiple models.

Appendix B

Models

Figure B.1: The armadillo mesh used as our testing model

Name: Armadillo

Number of vertices: 230854

Degrees of freedom: 692562

Sourced from: Stanford University Computer Graphics Laboratory

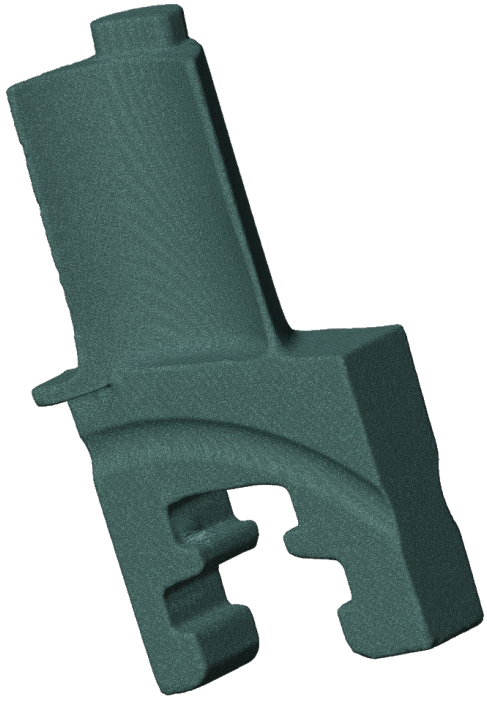
Name: Blade smooth

Number of vertices: 434815

Degrees of freedom: 1304466

Sourced from: Stanford University Computer Graphics Laboratory

Figure B.2: The blade cad mesh used as our testing model



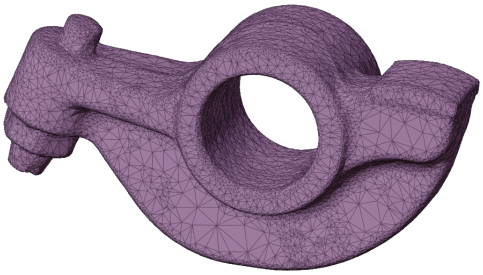
Name: Fandisk

Number of vertices: 12444

Degrees of freedom: 28065

Sourced from: CAD part Pratt & Whitney/Hughes Hoppe

Figure B.3: The fandisk mesh used as our testing model



Name: Rockerarm

Number of vertices: 18492

Degrees of freedom: 55476

Sourced from: INRIA

Figure B.4: The rockerarm mesh used as our testing model

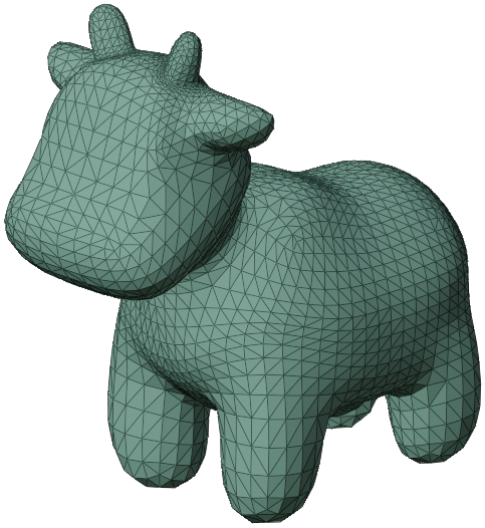
Name: Sphere

Number of vertices: 60827

Degrees of freedom: 182481

Sourced from: Created with Gmsh [Geuzaine, Christophe and Remacle, Jean-Francois]

Figure B.5: The sphere mesh used as our testing model



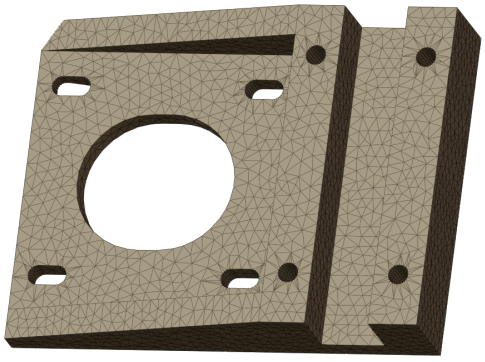
Name: Spot

Number of vertices: 9355

Degrees of freedom: 28065

Sourced from: Keenan Crane[Crane et al., 2013]

Figure B.6: The spot mesh used as our testing model



Name: Stepper mount

Number of vertices: 11020

Degrees of freedom: 33060

Sourced from: GrabCAD[a] user: Mudzaqi,name: Bracket Motor Stepper

[a] https://grabcad.com

Figure B.7: The stepper mount cad mesh used as our testing model